SpelBots 2006 RoboCup Technical Report

Ashley Johnson, Kina McCanns, Andrea Roberson, Ebony Smith, Andrew Williams

Artificial Intelligence, Bioinformatics, and Robotics Lab
Spelman College
350 Spelman Lane SW
Atlanta, GA 30308

Contents

I. Introduction

The Spelman College robotics team, SpelBots, has evolved into a multi-disciplinary academic environment. There are three students in the computer science department and one student from the mathematics department. The team is comprised of one returning member from the previous competition, Ebony Smith, who serves as the team lead. The remaining members of the team are new and have added another layer of enthusiasm combined with hard work.

The SpelBots team was notably challenged in all aspects of the RoboCup 2005 competition. This code from RoboCup 2005 serves as a baseline to begin enhancements for competitions in 2006. Utilizing the experience of RoboCup 2005, the SpelBots began improving on their previous code. The SpelBots have made close companions with each of the ERS-7 robots in the laboratory. Our focus has been to study diligently in efforts to improve all areas of our previous game.

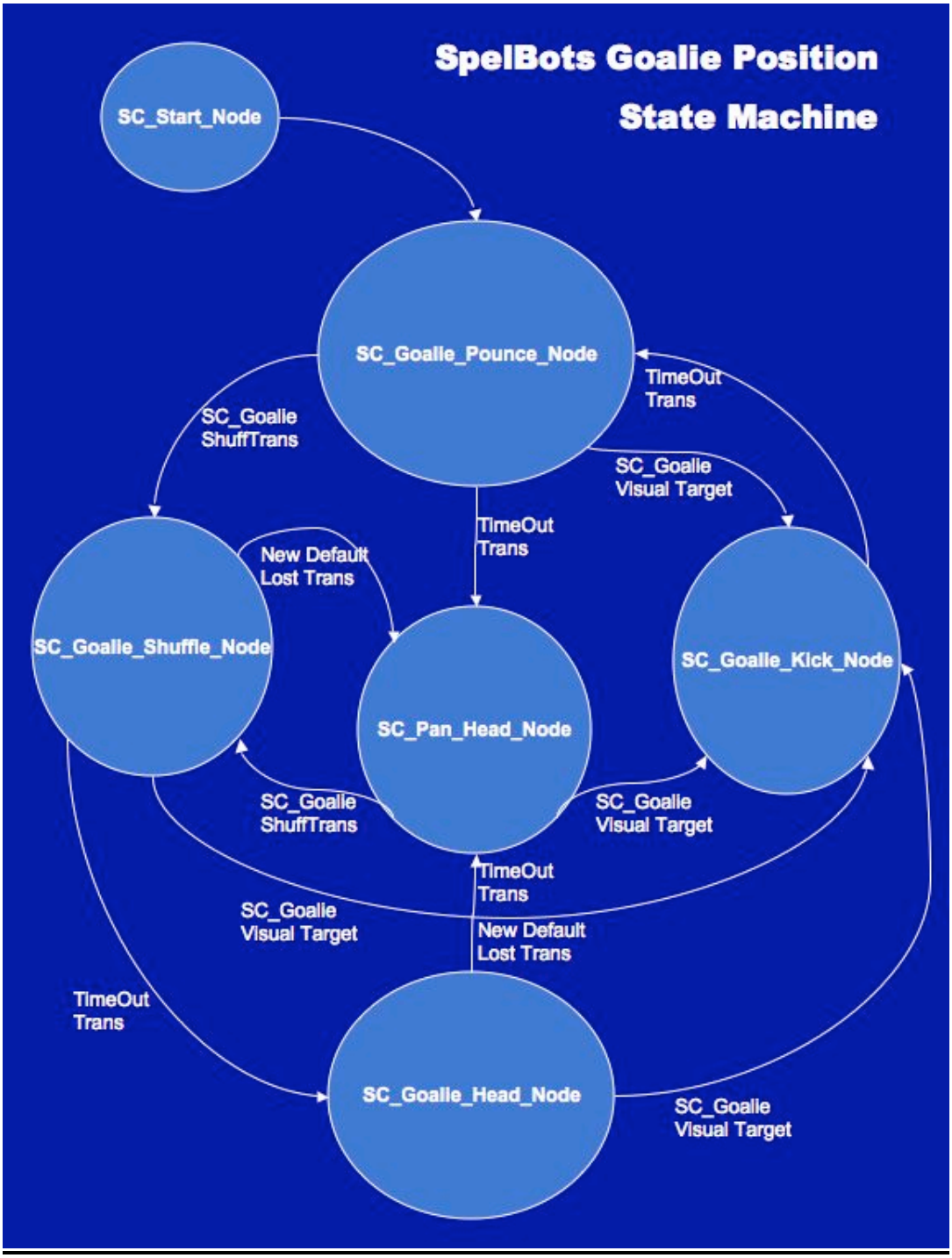
Similar to our code in 2005, the SpelBots did not use pre-existing soccer code from other RoboCup teams. The code developed is unique in that it uses the Tekkotsu Robot programming framework. The Tekkotsu project is an open source software project that makes use of other base models including earlier versions of CMVision and CMWalk and ROBOOP. The SpelBots continually strive to introduce our all-women undergraduate institution to the fascinating world of robotics and artificial intelligence. We look forward to competing again in RoboCup 2007 with more improvements in our team's vision, localization, and strategy.

This, second year competing, continues to be a great learning experience and allows the team to interact with the more veteran teams of RoboCup. The UNSW and UPenn teams were a great help this year during the competition. Their teams were very forthcoming with information and patient with our understanding of the smallest details.

Overall, the women on the Spelman College robotics team have incorporated sedulous study and a profound work ethic aimed at qualifying and being a worthy competitor for RoboCup 2006.

II. Goalie Design Document

For RoboCup 2006, the SpelBots team created several documents to support the goalie state machine. These documents include one main StateNode, several transitions, and other StateNodes that support special motions specific to the goalie. This document lists and describes these files with great detail.



Files Used

G. SC_Goalie_Lil_Shuff.h
    - Main goalie StateNode that implements goal state machine
H. GoalieWalkNode (.cc and .h)
    - Implements a walk that follows the ball horizontally within the goalie box.
I. GoalieHeadFollowNode (.cc and .h)
    - Implements a motion that allows the goalie to track the ball using only its head joints, specifically the pan joint.
J. SC_GoalieVisualTarget.h
    - Used to transition to into a block/kick when the ball is close enough to the goal.
K. SC_GoalieShuffTrans.h
    - Used to transition into a walk that follows the ball horizontally when the ball is sighted at a specified distance.
L. TimeOutTrans
    - Used to transition to another node after executing the current node for a specified amount of time.

A. SC_Goalie_Lil_Shuff.h

This document is the final layout of the goalie code that was implemented during RoboCup 2006 in Bremen, Germany by the SpelBots team. This section corresponds to the file entitled SC_Goalie_Lil_Shuff.h. After several different algorithms were tried and tested, this particular algorithm proved to be the best for the SpelBots team. The file is one very large Tekkotsu StateNode. Within the large StateNode, there are smaller StateNodes that handle specific tasks. This section explains in detail each of these smaller nodes and the reasoning behind them.

- **Name:** SC_Start_Node
    - **Purpose:** Specifies the first small node that will be executed in the StateNode
    - **Input Transitions:** Initialization of execution of file
    - **Output Transitions:** None
    - **Basic Algorithm:**
        1. Upon the start of the program, the start node signals the first node to be executed.

- **Name:** SC_Goalie_Pounce_Node
    - **Purpose:** Allows the robot to begin in a neutral stance to maximize field coverage. Also, this stance gives the robot a height that is suitable for transitioning into a powerful block/kick.
    - **Input Transitions:**

1. Start Node pointer
2. SC_Goalie_Kick_Node → TimeOutTrans: After the robot kicks the ball, it recovers by transitioning back to this node to regain a neutral position

- **Output Transitions:**
  1. TimeOutTrans: If robot does not recognize the ball within 1.5 seconds, then transition to SC_Pan_Head_Node.
  2. SC_Goalie_Shuffle_Node: If the ball is recognized near the goalie box and is not close enough to kick, then transition to SC_Goalie_Shuffle_Node.
  3. SC_GoalieVisualTarget: If the ball is recognized from a very close location, then transition to SC_Goalie_Kick_Node.
- **Basic Algorithm:**
  1. Move robot into pounce position.
  2. While doing so, continue looking for the ball.
  3. The Node is a MediumMotionSequenceNode
  4. Motion file used: goalsit.pos
  5. The motion file does not manipulate any of the robot's head joints, only its body joints.

- **Name:** SC_Pan_Head_Node
  - **Purpose:** Allows the robot to move its head in search of the ball. This motion sequence give the robot nearly a 155 degree view of the field.
  - **Input Transitions:**
    1. SC_Goalie_Pounce_Node → TimeOutTrans: After moving to a neutral stance, it then moves its head in search of the ball.
    2. SC_Goalie_Shuffle_Node → newDefaultLostTrans: While shuffling to follow the ball, if the robot looses sight of it, transition to this node to search for it again.
  - **Output Transitions:**
    1. SC_GoalieShuffTrans: If the ball is recognized near the goalie box and is not close enough to kick, then transition to SC_Goalie_Shuffle_Node to begin walking sideways to follow it.
    2. SC_GoalieVisualTarget: If the ball is recognized near the goalie box and is close enough to kick, then transition to SC_Goalie_Kick_Node
  - **Basic Algorithm:**
    1. Pan robot's head in motion sequence (up, right, down, left) that will maximize the robot's field of view
    2. The node is a LargeMotionSequenceNode.
    3. Motion file used: newpan.mot
    4. Posture files
       a. hdupleft.pos
       b. hdupmid.pos
       c. hduprit.pos
       d. hddwnrit.pos

e. hddwnmid.pos
f. hddwnlef.pos

- **Name:** SC_Goalie_Kick_Node
  - **Purpose:** Allow the robot to kick the ball when it comes close. It is a kick that also serves as a block.
  - **Input Transitions:**
    1. SC_Goalie_Pounce_Node → SC_GoalieVisualTarget: If the ball is spotted in close proximity while dog is going into the pounce position, the robot should kick it.
    2. SC_Pan_Head_Node → SC_ GoalieVisualTarget: If the ball is spotted in close proximity while dog is panning its head, the robot should kick it.
    3. SC_Goalie_Shuffle_Node → SC_ GoalieVisualTarget: If the ball is spotted in close proximity while dog is shuffling in chase of it, the robot should kick it
    4. SC_Goalie_Head_Node → SC_GoalieVisualTarget: : If the ball is spotted in close proximity while dog is following the ball with just it head, the robot should kick it
  - **Output Transitions:**
    1. SC_Time_Out_Transition: After the robot has blocked/kicked the ball for 3.5 seconds, it should go back to a neutral search position by transitioning to SC_Goalie_Pounce_Node
  - **Basic Algorithm:**
    1. A GroupNode is used to implement a kick.
    2. Within the GroupNode, this is one SmallMotionSequenceNode called kickball.
    3. The kickball node used the motion file ekogoal.mot.
    4. Posture files:
       a. FALLDWN.pos
       b. SPDARM.pos
       c. ARMBND.pos
       d. START.pos

- **Name:** SC_Goalie_Shuffle_Node
  - **Purpose:** Allows the robot to move horizontally in the same direction of the ball, while maintaining eyesight of the ball and staying within the goalie box.
  - **Input Transitions:**
    1. SC_Goalie_Pounce_Node → SC_GoalieShuffTrans: While the robot is moving into a neutral stance, if the ball is sighted at a distant location, the robot should begin moving horizontally to follow the ball's movement.

2. SC_Pan_Head_Node → SC_GoalieShuffTrans: While the robot is panning its head, if the ball is sighted at a distant location, the robot should begin moving horizontally to follow the ball's movement.

- **Output Transitions:**
  1. SC_GoalieVisualTarget: If the ball is spotted from a position close enough to kick it, transition to SC_Goalie_Kick_Node
  2. SC_TimeOutTrans: If the robot has been moving horizontally following the ball for more than 3 seconds, it should then begin tracking the ball's movement with only its head joints be transitioning to SC_Goalie_Head_Node.
  3. newDefaultLostTrans: If the robot looses sight of the ball during its shuffle, transition to SC_Pan_Head_Node
- **Basic Algorithm:**
  1. The robot moves horizontally based on the horizontal position if the ball.
  2. To implement the horizontal walk, a GoalieWalkNode is used.

- **Name:** SC_Goalie_Head_Node
  - **Purpose:** Allows the robot to follow the ball by using only its head joints. This motion allows the goalie to stay within the goalie box, but still track the ball's position. It reduces the chances of the goalie leaving its box and being unable to find its way back.
  - **Input Transitions:** SC_Goalie_Shuffle_Node → TimeOutTrans: After following the ball horizontally with its entire body for 3 seconds, the robot transitions to this node to avoid leaving the goalie box.
  - **Output Transitions:**
    1. SC_GoalieVisualTarget; If the ball is sighted in a close proximity, begin kicking/blocking by transitioning to SC_Goalie_Kick_Node.
    2. SC_Time_Out_Transition; If robot has been following the ball withits head for more than 2 seconds, transition to SC_Pan_Head_Node.
    3. newDefaultLostTrans: If the robot looses sight of the ball white following it with its head joints, transition to SC_Pan_Head_Node.
  - **Basic Algorithm:**
    1. Track the balls movement using only the head joints.
    2. Implemented with a GoalieHeadFollowNode

Transitions

- **Name:** SC_GoalieVisualTarget
  - **Purpose:** Fired when a specified object is recognized within the 200 vision threshold distance.

- **Name:** TimeOutTrans
    - **Purpose:** If the robot should remain in a node for a set amount of time, use this transition to limit that time and move on to the next node.

- **Name:** SC_ GoalieShuffTrans
    - **Purpose:** When the ball is sighted with the 300 vision threashold distance, the robot beginning moving its body horizontally in chase of the ball.

- **Name:** newDefaultLostTrans
    - **Purpose:** If an object was once in sight, but now its not, use this transition to determine the robot's next move. Built-in transition with StateNode files using the TimeOutTrans.h file.

B. GoalieWalkNode (.cc and .h)

The GoalieWalkNode files are based on the WalkToTargetNode files used by the attacker. This file is a state node for walking horizontally toward a visual target.

Constructors:

- The first constructor passes one parameter of the VisionObjectSourceID_t for the object that should be tracked by the robot.
- Similarly, the second constructor passes an instance name of the node as well as the VisionObjectSourceID_t for the object that should be tracked by the robot.

Algorithm:

In the DoStart of the .cc file, a headpointer_id and a walk_id are initialized using the motion manager, the HeadPointerMC, and the WalkMC. Also, a listener is added to the event router to listen for events corresponding to the VisionObjectSourceID_t.

The main action of the code is found in the processEvent function. When an event is thrown, first it is check to ensure that it is a vision object event and that it is a status event. Once both these cases are true, the horizontal and vertical values of the object's center are obtained. These values are used to set the values of the robot's head joints using the HeadPointerMC. After all of the head joints are set, the robot's head will point to the center of the specified vision object.

Once the robot's head is pointed toward the desired object (in our case, the ball), the code then uses the WalkMC to allow the robot to follow the ball with its body. Using the setTargetVelocity function, the robot moves horizontally based on the pan value of its head. This is done by setting the x and z values in the setTargetVelocity function to zero. The y value (which corresponds horizontal motion) is set to pan*100. This only happens

when the pan values is between -.05 and .05, other wise the robot does not move at all. The insures that one the robot has lined up vertically with the ball, it will cease horizontal movement with its body.

C. GoalieFollowHeadNode (.cc and .h)

The GoalieFollowHeadNode files are based on the GoalieWalkNode files. This file is a state node for following visual target using only the robots head joints.

Constructors:

- The first constructor passes one parameter of the VisionObjectSourceID_t for the object that should be tracked by the robot.
- Similarly, the second constructor passes an instance name of the node as well as the VisionObjectSourceID_t for the object that should be tracked by the robot.

Algorithm:

In the DoStart of the .cc file, a headpointer_id is initialized using the motion manager and the HeadPointerMC. Also, a listener is added to the event router to listen for events corresponding to the VisionObjectSourceID_t.

The main action of the code is found in the processEvent function. When an event is thrown, first it is check to ensure that it is a vision object event and that it is a status event. Once both these cases are true, the horizontal and vertical values of the object's center are obtained. These values are used to set the values of the robot's head joints using the HeadPointerMC. After all of the head joints are set, the robot's head will point to the center of the specified vision object.

D. SC_GoalieVisualTarget,h

The SC_GoalieVisualTarget.h file is based on the VisualTargetCloseTransition.h file. This file is a transition that fires when the ball is close enough to the goalie for block/kick.

Constructors:

- The first parameter of the first constructor is the name of a destination node to with the transition lead. The second parameter is a source id for a VisionObjectSourceID_t. The last parameter is the distance threshold that

specifies the vision object should have an IRDistOffset of 200 or less when the transition is thrown.
- Similarly, the second constructor passes all of the same information as the pervious one, but is adds a specific class name to the destination node.

Algorithm:

In the processEvent function, first a VisionObjectEvent is defined. When a VisionObjectEvent occurs, the x and y values for the object's center are obtained. If the IRDistOffset of the object's center is less than the specified distance threshold of 200, the transition is fired.

E. SC_GoalieShuffTrans,h

The SC_GoalieShuffTrans.h file is based on the VisualTargetCloseTransition.h file. This file is a transition that fires when the ball is sighted at a distance to far away to kick.

Constructors:
- The first parameter of the first constructor is the name of a destination node to with the transition lead. The second parameter is a source id for a VisionObjectSourceID_t. The last parameter is the distance threshold that specifies the vision object should have an IRDistOffset of 300 or less when the transition is thrown.
- Similarly, the second constructor passes all of the same information as the pervious one, but is adds a specific class name to the destination node.

Algorithm:

In the processEvent function, first a VisionObjectEvent is defined. When a VisionObjectEvent occurs, the x and y values for the object's center are obtained. If the IRDistOffset of the object's center is less than the specified distance threshold of 300, the transition is fired.

F. TimeOutTrans,h

The TimeOutTrans.h file is a Tekkotsu Transition that is fired after a StateNode has executed for a specified about of time.
Constructors:

- The first constructor passes two parameters. The first parameter is the name of the destination node for the transition. The second is the delay for transition in milliseconds.
- The second constructor has the same two parameters of the previous constructor as well as a third parameter of an event generator id. If any events of that type are received, the timer for the transition resets.
- The third and fourth constructors are similar to the second, but they allow more specific information regarding the event to be passed as well in additional parameters.

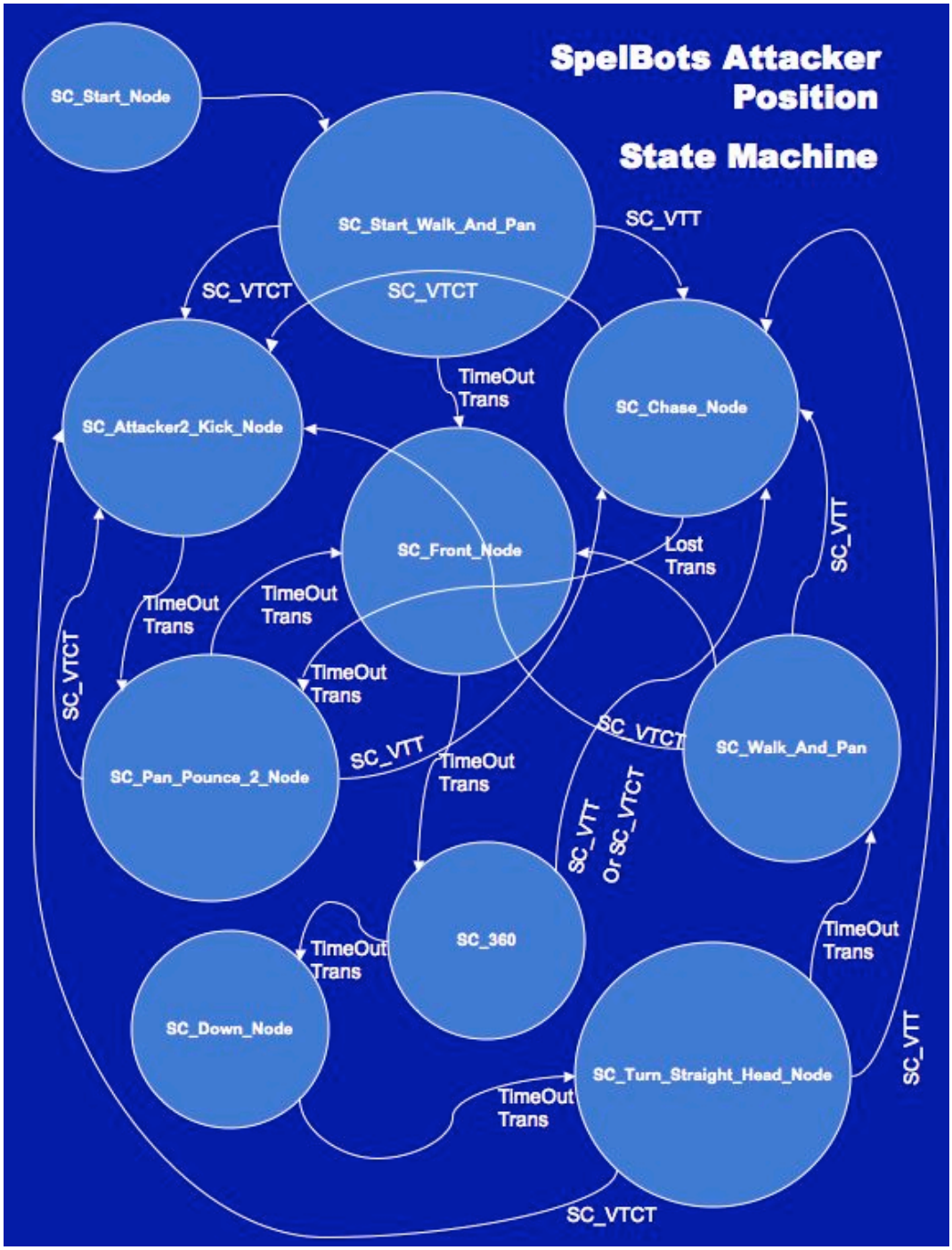
Algorithm:

In the DoStart, a switch statement is used is add listeners of specific events based on which of the constructors is used. At the end of the DoStart, the timer is reset.

The resetTimer function throws an event that sets the timer to zero. In the processEvent function, if an event is thrown by the timer (meaning the timer has expired), the transition is fired. Else, the time is reset.

III. Attacker Design Document

For RoboCup 2006, the SpelBots team created several documents to support the attacker state machine. These documents include one main StateNode, several transitions, and other StateNodes that support special motions specific to the attacker. This document lists and describes these files with great detail.



Files Used

A. SC_Attacker2.h
    - Main attacker StateNode that implements attacker state machine

A. SC_Attacker2.h

This document is the final layout of the attacker code that was implemented during RoboCup 2006 in Bremen, Germany by the SpelBots team. This section corresponds to the file entitled SC_Attacker2.h. After several different algorithms were tried and tested, this particular algorithm proved to be the best for the SpelBots team. The file is one very large Tekkotsu StateNode. Within the large StateNode, there are smaller StateNodes that handle specific tasks. This section explains in detail each of these smaller nodes and the reasoning behind them.

- **Name:** SC_Start_Node
    - **Purpose:** Specifies the first node that will be executed in the behavior
    - **Input Transitions:** Initialization of execution of behavior
    - **Output Transitions:** None
    - **Basic Algorithm:**
        2. Upon the start of the program, the start node will point to the first node to be executed.

- **Name:** SC_Start_Walk_And_Pan
    - **Purpose:** Allows the attack to start out by moving forward while looking for the ball. At the beginning of any kickoff, the ball will always be in front of the attackers. By moving forward first, the robot will always begin by moving closer to the ball.
    - **Input Transitions:** Initialization of execution of behavior
    - **Output Transitions:**
        1. TimeOutTrans: If robot does not recognize the ball within 3.5 seconds, then transition to SC_frontNode.
        2. VisualTargetCloseTrans: If the ball is recognized and is close enough to kick, then transition to SC_Attacker2_Kick_Node
        3. VisualTargetTrans: If the ball is recognized from a very distant location, then transition to SC_ Chase_Node.
    - **Basic Algorithm:**
        1. Upon the start of the program, the start node will point to this node.
        2. This node should be a group node
        3. The first node in the group should be a node that allows the robot to walk forward.
        4. The second node should be a node that incorporates a head pan.
        5. Be careful that the head pan maximizes coverage of the field and does not look over the ball too much.

- **Name:** SC_Pan_Pounce_2_Node
    - **Purpose:** Allows the robot to begin in a static search position with maximum field view after ball has been kicked at least once.
    - **Input Transitions:**
        1. SC_Chase_Node → newDefaultLostTrans
        6. SC_Attacker2_Kick_Node → TimeOutTrans
    - **Output Transitions:**
        3. TimeOutTrans: If robot does not recognize the ball within 2.5 seconds, then transition to SC_frontNode.
        4. VisualTargetCloseTrans: If the ball is recognized and it is close enough to kick, then transition to SC_Attacker2_Kick_Node
        5. VisualTargetTrans: If the ball is recognized from a very distant location, then transition to SC_Chase_Node.
    - **Basic Algorithm:**
        5. Move robot into pounce position.
        6. Pan robot's head in motion sequence (up, right, down, left) that will maximize the robot's field of view
        7. If ball is seen up close then kick it.
        8. If ball is seen at a distance, then run towards it.
        9. If ball isn't seen within 2.5 seconds, then turn in a circle and search.

- **Name:** SC_Attacker2_Kick_Node
    - **Purpose:** Allow the robot to kick the ball when it comes close. This kick will be different from the goalie kick because it should give precedence to power, rather than blocking.
    - **Input Transitions:**
        5. SC_Pan_Pounce_2_Node → VisualTargetCloseTrans
        6. SC_Start_Walk_And_Pan → VisualTargetCloseTrans
        7. SC_Chase_Node → VisualTargetCloseTrans
        8. SC_360 → VisualTargetCloseTrans
    - **Output Transitions:**
        2. TimeOutTrans: After the robot has finished kicking the ball for 1 second, it should go back to a neutral search position by transitioning to SC_Pan_Pounce_2_Node

    - **Basic Algorithm:**
        5. A Motion sequence should be used to implement a kick.
        6. The postures that make up the motion sequence should be gradual changes in position for the robot's joints
        7. The kick be very powerful.

- **Name:** SC_Chase_Node
    - **Purpose:** Allows the robot to run to the ball when it has been spotted from afar.
    - **Input Transitions:**
        1. SC_Pan_Pounce_2_Node → VisualTargetTrans
        2. SC_Start_Walk_And_Pan → VisualTargetTrans
        3. SC_Turn_Straight_Head_Node → VisualTargetTrans
    - **Output Transitions:**
        4. VisualTargetCloseTrans: If the ball is spotted from a position close enough to kick it, transition to SC_Attacker2_Kick_Node.
        5. newDefaultLostTrans: If the robot does not see the ball any more, transition to SC_Pan_Pounce_2_Node.
    - **Basic Algorithm:**
        3. The robot should walk towards the ball while maintaining the ball within its view.
        4. Refer to Explore Group and HeadPointerNode

- **Name:** SC_360
    - **Purpose:** Allows the robot to turn in a complete circle in search of the ball.
    - **Input Transitions:** SC_frontNode → TimeOutTrans
    - **Output Transitions:**
        4. VisualTargetTrans; If the ball is spotted from afar, then robot should transition to SC_Chase_Node.
        5. VisualTargetCloseTrans: If the ball is spotted from a position close enough to kick it, transition to SC_Attacker2_Kick_Node
        6. TimeOutTrans: If the robot does not see the ball within 2.5 seconds, transition to SC_downNode.
    - **Basic Algorithm:**
        3. Turn in Place until ball is recognized
        4. Once ball is recognized from a distance, walk towards it.
        5. If the ball is not recognized, keep turning in place with a different head position.
        6. Ideally, the ball should be spotted after turning for so long.

- **Name:** SC_Turn_Straight_Head_Node
    - **Purpose:**
    - **Input Transitions:** SC_downNode → TimeOutTrans
    - **Output Transitions:**
        1. VisualTargetTrans; If the ball is spotted from afar, then robot should transition to SC_Chase_Node.
        2. VisualTargetCloseTrans: If the ball is spotted from a position close enough to kick it, transition to SC_Attacker2_Kick_Node.
        3. TimeOutTrans: If the ball is not spotted within 6 seconds, transition to SC_Walk_And_Pan.
    - **Basic Algorithm:**

1. Turn in Place until ball is recognized
2. Once ball is recognized from a distance, walk towards it.
3. Once ball is recognized from up close, kick it.
4. If the ball is not recognized, walk to a different location and look for the ball.

- **Name:** SC_Walk_And_Pan
  - **Purpose:** Allows the robot to walk and search for the ball.
  - **Input Transitions:**
    1. SC_Turn_Straight_Head_Node → TimeOutTrans
  - **Output Transitions:**
    1. VisualTargetCloseTrans: If the ball is spotted from a position close enough to kick it, transition to SC_Attacker2_Kick_Node.
    2. VisualTargetTrans: If the ball is spotted from afar, then robot should transition to SC_Chase_Node.
    3. TimeOutTrans: If the ball is not spotted within 3 seconds, transition to SC_frontNode.
  - **Basic Algorithm:**
    1. The robot should walk forward, while panning its head.
    2. Once ball is recognized from a distance, the robot will chase the ball.
    3. If the ball is really close to the robot, the robot will kick the ball.
    4. If the ball is not seen at all within 3 seconds, then the robot will turn in place searching for the ball.

- **Name:** SC_downNode
  - **Purpose:** Allows the robot to view the ball if it is in front of it.
  - **Input Transitions:**
    1. SC_360 → TimeOutTrans
  - **Output Transitions:**
    1. TimeOutTrans: After half a second, transition to the SC_Turn_Straight_Head_Node.
  - **Basic Algorithm:**
    1. Robot's head should point down to search the field for the ball up close.
    2. After half a second, the robot will turn in a circle.

- **Name:** SC_frontNode
  - **Purpose:** Allows the robot to view the ball if it is in the distance.
  - **Input Transitions:**
    1. SC_Pan_Pounce_2_Node → TimeOutTrans
    2. SC_Start_Walk_And_Pan → TimeOutTrans
    3. SC_Walk_And_Pan → TimeOutTrans
  - **Output Transitions:**
    1. TimeOutTrans: : After half a second, transition to the SC_360.
  - **Basic Algorithm:**

1. Robot's head should point up to search the field for the ball in the distance.
2. After half a second, the robot will turn in a circle.

Transitions

- **Name:** VisualTargetTrans
  - **Purpose:** If an object is spotted recognized, no matter how far away it may appear, use this transition.

- **Name:** VisualTargetCloseTrans
  - **Purpose:** If the ball is spotted from a very close position.

- **Name:** TimeOutTrans
  - **Purpose:** If the robot should remain in a node for a set amount of time, use this transition to limit that time and move on to the next node.

- **Name:** newDefaultLostTrans
  - **Purpose:** If an object was once in sight, but now its not, use this transition to determine the robot's next move. Built-in transition with StateNode files using the TimeOutTrans.h file.

IV. Vision Documentation: Blob Detection

To complement the EasyTrain vision tool of Tekkotsu, the SpelBots used blob detection algorithms to identify desirable field attributes. The objects identified using these algorithms are the blue goal, the yellow goal, and the orange ball. The following is a list of the files used to detect blobs that correspond to these items along with detailed description of how they work.

D. SC_FindOrangeBallEvent.h
    - Find orange blods and identifies of one them as the ball, it possible.
E. SC_FindOrangeBlueEvent.h
    - Finds the orange ball and checks to see if it is in front of the blue goal.
F. SC_FindOrangeYellowEvent.h
    - Finde the orange ball and checks to see if it is in front of the yellow goal.

A. SC_FindOrangeBallEvent.h

The SC_FindOrangeBallEvent.h file is a VisualRoutinesBehavior that attempts to identify the orange ball by locating all of the orange blobs within a frame and using their width, height, and area to determine whether one of them is the ball.

In the DoStart, a listener for visual region event is added to the event router. The variable *found* is set to zero. In the processEvent function, first *found* is set to zero to indicate that the event was not found whenever a visRegion event is posted. Next, all of the blobs are obtained from the current camera. SHAPEVEC functions are then used to get all of the orange blobs then the yellow blobs from the blob data obtained when all of the blobs in the frame were identified.

After all of the yellow and orange blobs are identified, each orange blob is checked one by one to see whether it is the orange ball. This is done by first getting the area of the blob. Then the area is check to see whether it is between 70 and 17000. If the blob has an area less than 70, it is too small to be the ball. If the ball's area is over 17000, it is too large to be the ball. Next, the height and width of blob is obtained. The height and width is used to determine if the blob is roughly a square by checking to see is two values differ by less than 20%. If a blob pasts all of these tests, than more than likely the orange blob is a ball. An VisionObjectEvent is then throw to signal that the orange ball has been found.

B. SC_FindOrangeBlueEvent.h

The SC_FindOrangeBlueEvent.h file is a VisualRoutinesBehavior that attempts to identify the orange ball by locating all of the orange blobs within a frame and using their width, height, and area to determine whether one of them is the ball. It then analyzes all of the blue blobs in the frame to determine whether one of them is the blue goal. If the blue goal is found, then the file checks to see if the orange ball is located in front of the blue blob determined to be the goal.

In the DoStart, a listener for visual region event is added to the event router. The variables *found* is set to zero. In the processEvent function, first *found* and *foundBlue* are set to zero to indicate that the event was not found whenever a visRegion event is posted. Next, all of the blobs are obtained from the current camera. SHAPEVEC functions are then used get all of the orange blobs then all the blue blobs from the blob data obtained when all of the blobs in the frame were identified.

After all of the blue and orange blobs are identified, each blue blob is checked one by one to see whether it is the blue goal. This is done by first getting the area of each individual blob. The area is then check to see if is greater than 800 and larger than all of the other blue blobs that were previously analyzed. If the blob pasts these tests, the x- and y-coordinates of the center of the blob are obtained. The maxAreaBlue (holds the value of the area of the largest blob found) variable is updated to reflect the area of the current blob. The *foundBlue* variable is set to 1 to indicated that the blue goal has been found.

Next, the orange blobs in the camera frame are analyzed to determine whether one of them is the orange ball. This is done by first getting the area of each blob. Then the area is check to see whether it is between 70 and 17000. If the blob has an area less than 70, it is too small to be the ball. If the ball's area is over 17000, it is too large to be the ball. If the blob pasts these tests, the x- and y-coordinates of the center of the blob are obtained. Next, the height and width of blob is obtained. The height and width is used to determine if the blob is roughly a square by checking to see is two values differ by less than 20%. If a blob pasts all of these tests, than more than likely the orange blob is a ball. The *found* variable is set to 1 to indicated that the orange ball has been found.

Lastly, if the ball and the blue goal are found, a VisionObjectEvent is trown to indicate this occurrence. Or, if only the ball was found, then a different VisionObjectEvent is thrown to indicate this case. If the ball was not found at all, the *found* and the *foundBlue* variables are both set to zero.

C. SC_FindOrangeYellowEvent.h

The SC_FindOrangeYellowEvent.h file is a VisualRoutinesBehavior that attempts to identify the orange ball by locating all of the orange blobs within a frame and using their width, height, and area to determine whether one of them is the ball. It then analyzes all of the yellow blobs in the frame to determine whether one of them is the yellow goal. If the yellow goal is found, then the file checks to see if the orange ball is located in front of the yellow blob determined to be the goal.

In the DoStart, a listener for visual region event is added to the event router. The variables *found* is set to zero. In the processEvent function, first *found* and *foundYellow* are set to zero to indicate that the event was not found whenever a visRegion event is posted. Next, all of the blobs are obtained from the current camera. SHAPEVEC functions are then used get all of the orange blobs then all the yellow blobs from the blob data obtained when all of the blobs in the frame were identified.

After all of the yellow and orange blobs are identified, each yellow blob is checked one by one to see whether it is the yellow goal. This is done by first getting the area of each individual blob. The area is then check to see if is greater than 800 and larger than all of the other yellow blobs that were previously analyzed. If the blob pasts these tests, the x- and y-coordinates of the center of the blob are obtained. The maxAreaYellow (holds the value of the area of the largest blob found) variable is updated to reflect the area of the current blob. The *foundYellow* variable is set to 1 to indicated that the yellow goal has been found.

Next, the orange blobs in the camera frame are analyzed to determine whether one of them is the orange ball. This is done by first getting the area of each blob. Then the area is check to see whether it is between 70 and 17000. If the blob has an area less than 70, it is too small to be the ball. If the ball's area is over 17000, it is too large to be the ball. If the blob pasts these tests, the x- and y-coordinates of the center of the blob are obtained. Next, the height and width of blob is obtained. The height and width is used to determine if the blob is roughly a square by checking to see is two values differ by less than 20%. If a blob pasts all of these tests, than more than likely the orange blob is a ball. The *found* variable is set to 1 to indicated that the orange ball has been found.

Lastly, if the ball and the yellow goal are found, a VisionObjectEvent is trown to indicate this occurrence. Or, if only the ball was found, then a different VisionObjectEvent is thrown to indicate this case. If the ball was not found at all, the *found* and the *foundYellow* variables are both set to zero.

V. Motion and Locomotion Design

The term locomotion refers to the robots' ability to move around the field. The term motion refers to the robots' actions in relation to the soccer game.

The motions and locomotion movements for the dogs will be as follows:

Locomotion:

- Pan head Motions

Motions:

Goalie:
- Two-leg Block
- Goalie Kick

Attackers:
- Grab
- Shoulder Kick
- Head Kick
- Chest Kick
- Power Kick

These motions and loco motions are made using a Tekkotsu application called ControllerGUI. ControllerGUI is used to communicate with the AIBO through wireless signals. Using this tool, positions can be created and saved as position files (pos) by manually positioning the dog's joints while it rests in emergency stop. Motion sequences are created through coding. The motion sequence file (mot) calls the necessary position files to be loaded after specified times.

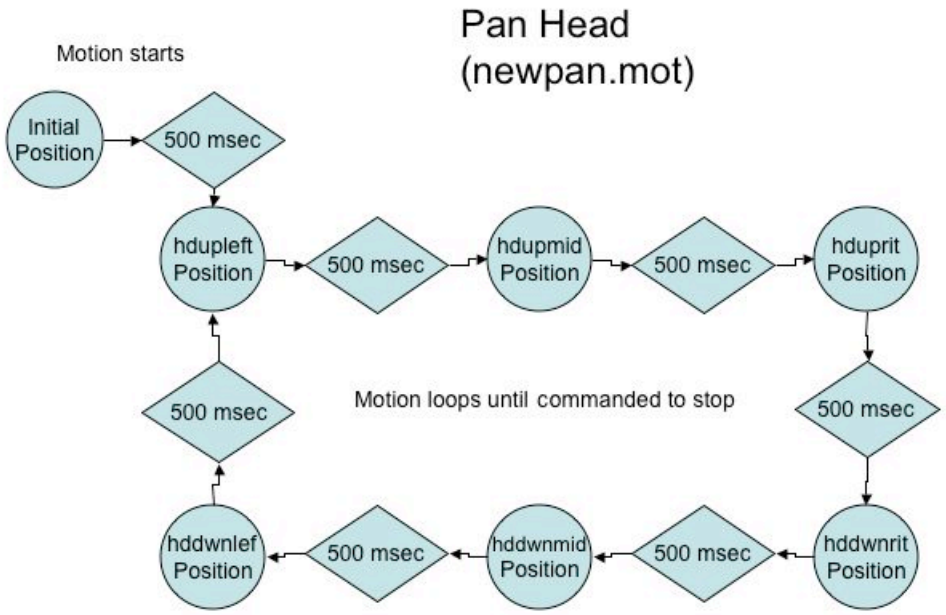
The following are the implementation plans for the above motions and loco motions:

- **Name:** Pan head
    - **File Name:** newpan.mot, turnpan.mot, begnattk.mot
    - **Purpose:** Every dog needs the ability to move its head around to search for the ball or the goal.
    - **Description:**
        - (newpan.mot) The head will turn starting from a central position. Then it will move to the left or right, down, to the left or right, up, then back to that central position. The head will move in a circular motion.
        - (turnpan.mot) As the dog turns, it keeps its head in a central position. After it has made a complete turn, it lowers its head then makes another complete turn.

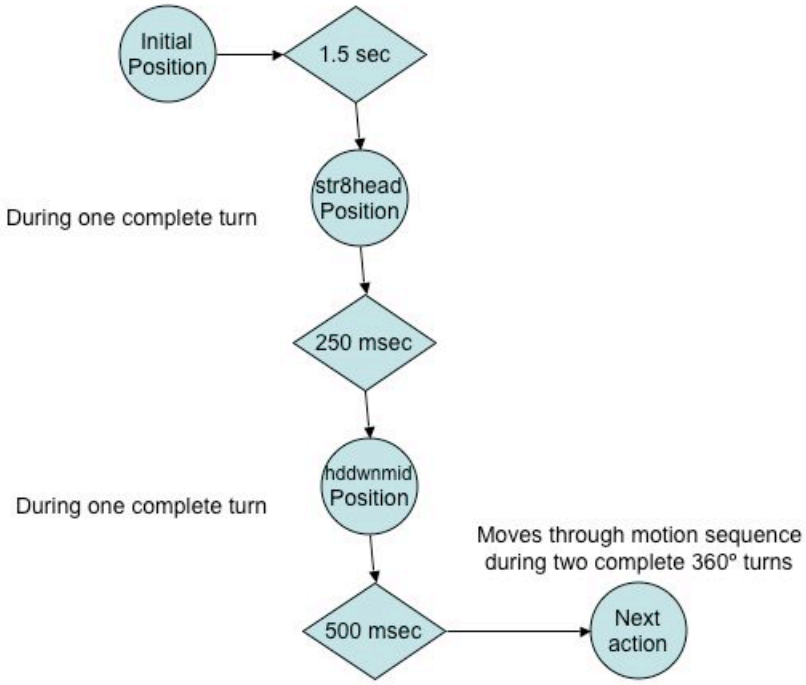
- (begnattk.mot) As the dog walks forward, it moves its head back and forth with it's neck at a central position to allow the dog to see a wide range of distances.

The pan heads will move in this fashion continuously until it is stopped by a condition programmed into the code of the behavior.

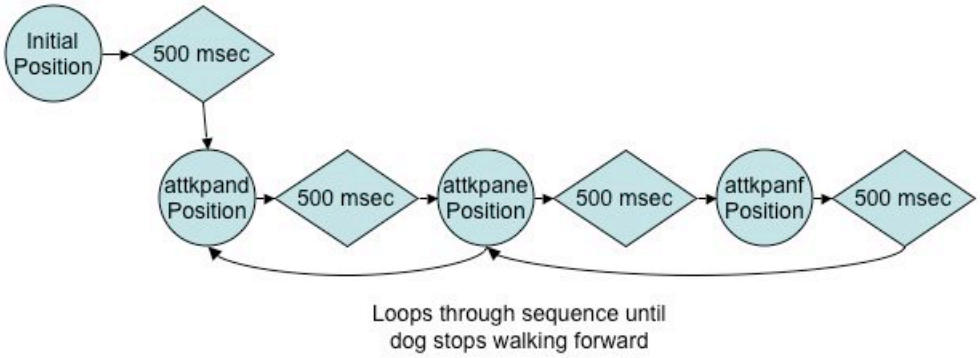
- **Method:** This motion will be composed of a position motion sequence.
- **Transitions:**
  - (newpan.mot) The times between the loading of every position file from the initial position to the last position is 500 msec.
  - (turnpan.mot) The times between the loading of each position file are 1.5 sec from the initial position to the srt8head.pos position, 250 msec from the str8head.pos position to the hddwnmid.pos position, and 500 msec from the hddwnmid.pos position to the position contained the node following the pan head or back to the str8head.pos position.
  - (begnattk.mot) The times between the loading of every position file from the initial position to the last position is 500 msec.
- **Positions:**
  - [newpan.mot] (6) hdupleft.pos, hdupmid.pos, hduprit.pos, hddwnrit.pos, hddwnmid.pos, hddwnlef.pos
  - [turnpan.mot] (2) str8head.pos, hddwnmid.pos
  - [begnattk.mot] (3) attkpand.pos, attkpane.pos, attkpanf.pos
- **Diagram:**



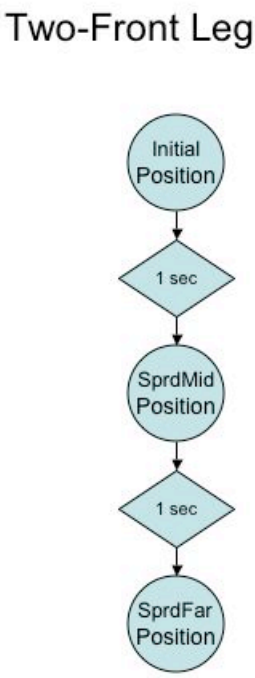
Pan Head
(turnpan.mot)



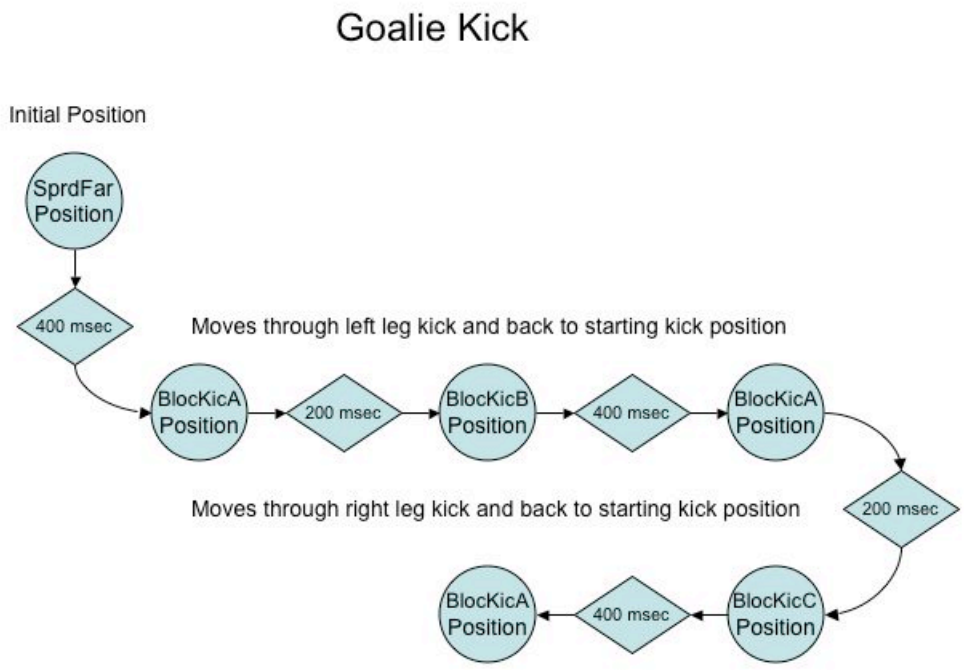
Pan Head
(begnattk.mot)



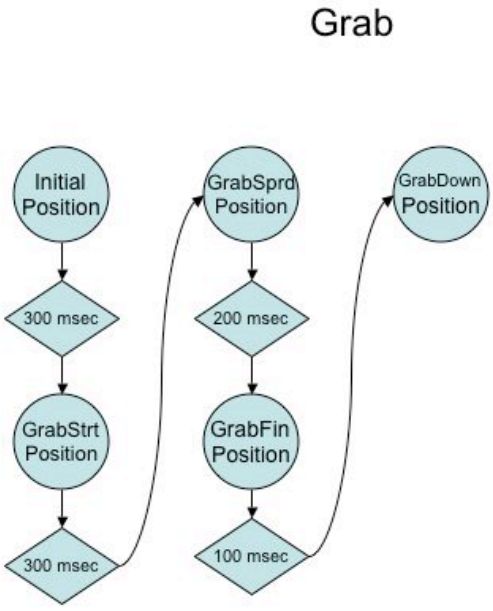
- **Name:** Two-Front Leg Block
  - **File Name:** goalblok.mot
  - **Purpose:** The goalie will use this block to cover as much of the area of the goal possible without violation of the blocking rules by stretching out both of its legs.
  - **Description:** When the goalie sees the ball close to its body, the dog will extend its front legs out as fast and as far as possible. This motion has already been designed in the past and will be tweaked if needed.
  - **Method:** This motion will be composed of a position motion sequence.
  - **Transitions:** the times between the loading of each position file are 1 second from the initial position to the Sprdmid.pos position, and 1 second from the SprdMid.pos position to the SprdFar.pos position.
  - **Positions:** (2) SprdMid.pos, SprdFar.pos
  - **Diagram:**



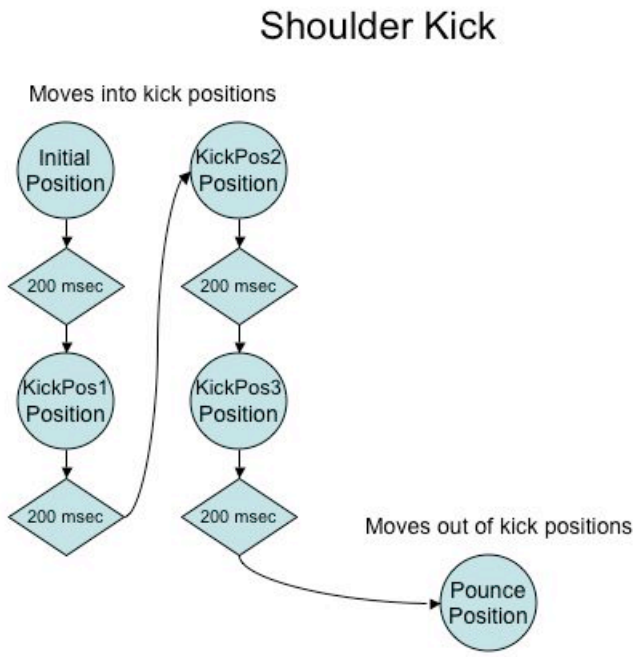
- **Name:** Goalie Kick
  - **File Name:** SC_Gkick.mot
  - **Purpose:** The goalie will use this kick to push the ball away from the goal after it is blocked in order to make scoring more difficult for the opposing team.
  - **Description:** After the dog gets into the two-front leg block position, the dog will move its left leg forward and back and then move its right leg forward and back. Moving both of the legs regardless of the position of the ball will allow the dog to hit the ball away without having to recognize which side of its body the ball is on. This kick can only be used when the dog is in the two-front leg block position.
  - **Method:** This motion will be composed of a position motion sequence.
  - **Transitions:** The times between the loading of each position file are 400 msec from the initial position to the BlocKicA.pos position, 200 msec from the BlocKicA.pos position to the BlocKicB.pos position, 400 msec from the BlocKicB.pos position to the BlocKicA.pos position, 200 msec from the BlocKicA.pos position to the BlocKicC.pos position, and 400 msec from the BlocKicC.pos position to the BlocKicA.pos position.
  - **Positions:** (3) BlocKicA.pos, BlocKicB.pos, BlocKicC.pos
  - **Diagram:**



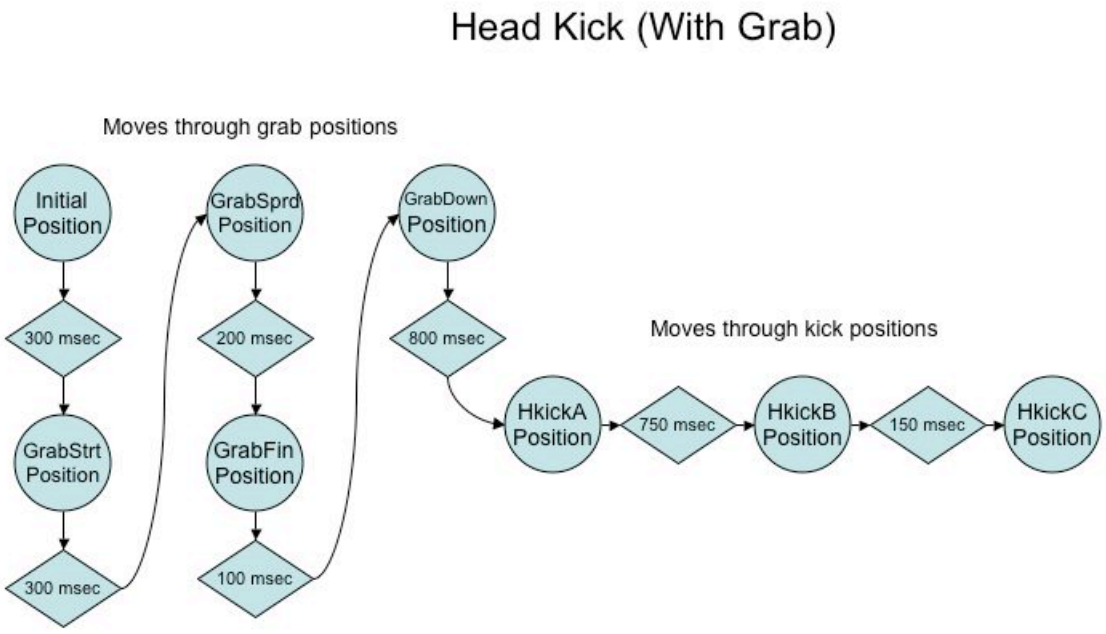
- **Name:** Grab
  - **File Name:** GrabA.mot
  - **Purpose:** This attacker can use this move to grab the ball before kicking it.
  - **Description:** Dog will trap the ball between its to front arms. The it will lower its head and open its mouth to gold onto the ball.
  - **Method:** This motion will be composed of a position motion sequence.
  - **Transitions:** The times between the loading of each file are 300 msec from the initial position to the GrabStrt.pos position, 300 msec from the GrabStrt.pos position to the GrabSprd.pos position, 200 msec from the GrabSprd.pos position to the GrabDown.pos position, and 100 msec from the GrabDown.pos position to the GrabFin.pos position.
  - **Positions:** (4) GrabStrt.pos, GrabSprd.pos, GrabDown.pos, GrabFin.pos
  - **Diagram:**



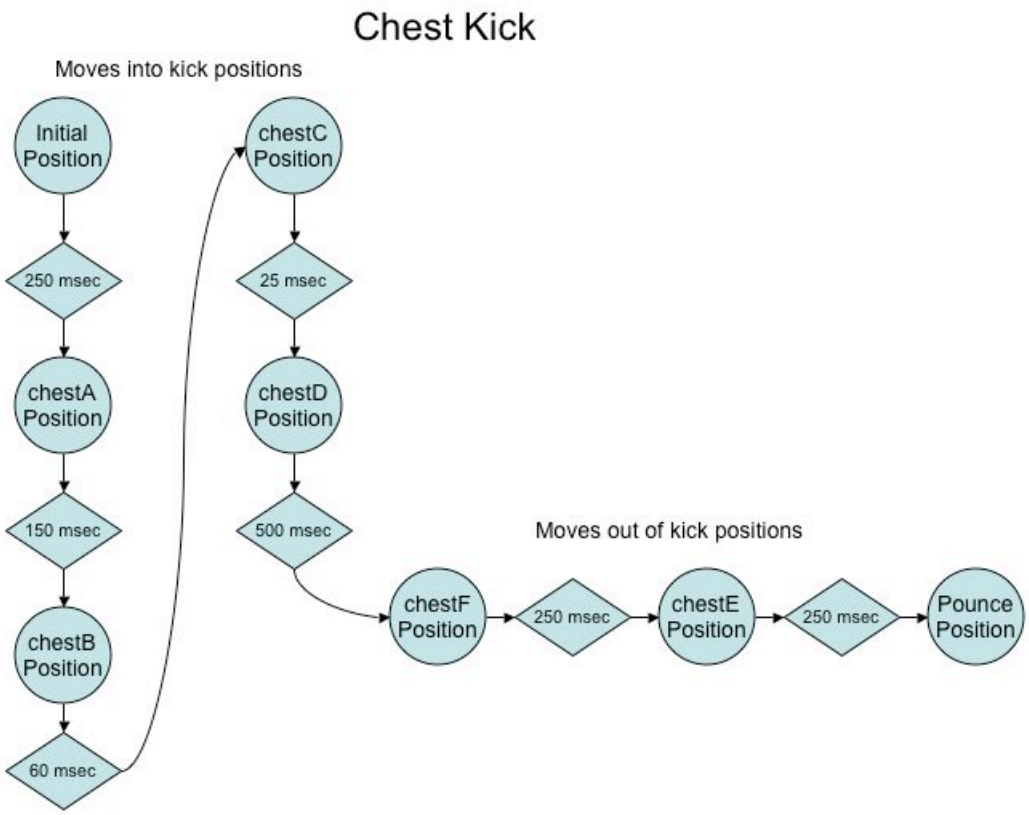
- **Name:** Shoulder Kick
  - **File Name:** SC_Shkic.mot
  - **Purpose:** This kick will push the ball farther down the field in a slightly random direction.
  - **Description:** The dog lunges forward and uses her head and arm to push the ball. The hits are random depending of the placement of the ball. This kick is also usually very powerful.
  - **Method:** This motion will be composed of a position motion sequence.
  - **Transitions:** The times between the loading of position each file are 200 msec from the initial position to the KickPos1.pos position, 200 msec from the KickPos1.pos position to the KickPos2.pos position, 200 msec from the KickPos2.pos to the KickPos3.pos, and 200 msec from the KickPos3.pos position to the pounce.pos position.
  - **Positions:** (4) KickPos1.pos, KickPos2.pos, KickPos3.pos, pounce.pos
  - **Diagram:**



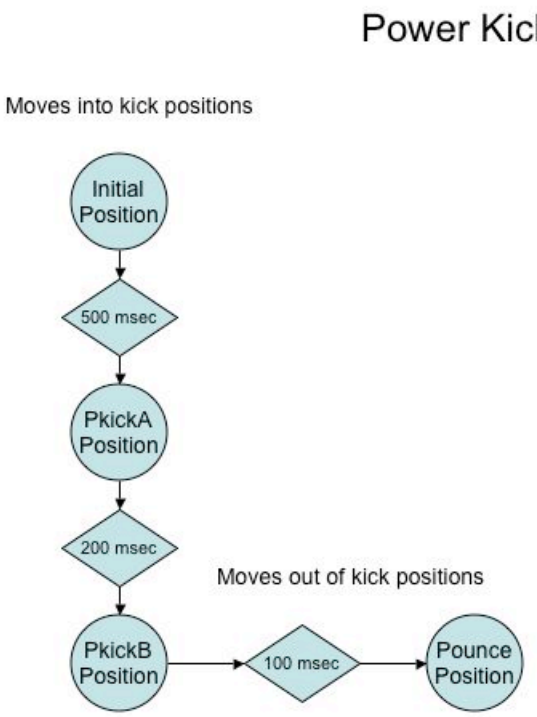
- **Name:** Head Kick (with grab)
    - **File Name:** SC_Hkick.mot
    - **Purpose:** This kick provides the dog with a straight and accurate kick that can be used when aiming towards the goal.
    - **Description:** The dog traps the ball using its mouth and arms. Then it lifts its head and uses its open mouth to push the ball away.
    - **Method:** The motion will be composed of added positions to the grab motion sequence.
    - **Transitions:** The times between the loading of each position file are 3 msec from the initial position to the GrabStrt.pos position, 3 msec from the GrabStrt.pos position to the GrabSprd.pos position, 200 msec from the GrabSprd.pos position to the GrabDown.pos position, 100 msec from the GrabDown.pos position to the GrabFin.pos position, and 800 msec from the GrabFin.pos position to the HkickA.pos position. The HkickA.pos position file transitions from the grab to the kick. The times between the position files for the actual kick are 750 msec between the HkickA.pos position and the HkickB.pos position, and 150 msec from the HkickB.pos position to the HkickC.pos position.
    - **Positions:** (7) GrabStrt.pos, GrabSprd.pos, GrabDown.pos, GrabFin.pos, HkickA.pos, HkickB.pos, HkickC.pos
    - **Diagram:**



- **Name:** Chest Kick
  - **File Name:** SC_Ckick.mot
  - **Purpose:** The chest kick will provide the dogs with a hard kick to get the ball across the field.
  - **Description:** This kick will be more common for all the dogs. When the ball has been positioned in front of the dog, the dog will extend its legs back supporting its weight on its paws. Then the dog will simultaneously roll its front legs and back legs back causing the dog to lunge forward to push the ball. The direction the ball is hit in is random and depends on the placement of the ball.
  - **Method:** This motion will be composed of a position motion sequence.
  - **Transitions:** The times between the loading of each position file are 250 msec from the initial position to the chestA.pos position, 150 msec from the chestA.pos position to the chestB.pos, 60 msec from the chestB.pos position to the chestC.pos position, 25 msec from the chestC.pos position to the chestD.pos position, 500 msec from the chestD.pos position to the chestE.pos position, 250 msec from the chestE.pos position to the chestF.pos position, and 250 from the chestF.pos position to the pounce.pos position.
  - **Positions:** (7) chestA.pos, chestB.pos, chestC.pos, chestD.pos, chestE.pos, chestF.pos, pounce.pos
  - **Diagram:**



- **Name:** Power Kick
  - **File Name:** SC_Pkickmot
  - **Purpose:** This kick is just an idea for an extremely hard kick that can be used to get the ball all the way across the field by the goalie or used by the attacker during kick off or a penalty kick.
  - **Description:** The dog lunges forward and uses it's body and head to butt the ball forward. The kick is dangerous because it forcibly pushes its head forward using its body.
  - **Method:** This motion will be composed of a position motion sequence.
  - **Transitions:** The times between the loading of each position file are 500 msec from the initial position to the pounce.pos position, 200 msec from the pounce.pos position to the PkickA.pos position, and 100 msec from the PkickA.pos position to the PkickB.pos position.
  - **Positions:** (3) PkickA.pos, PkickB.pos, pounce.pos
  - **Diagram:**



VI. Vision Calibration

Using the EasyTrain Vision Tool

Vision is one of, if not the most, important aspect of the RoboCup soccer competition. The AIBO robots can only rely on vision to understand and utilize their environment on the field. EasyTrain is a java application provided by Tekkotsu to create low-level vision segmentations. This application defines objects as having a specific, solid color without the distinction of shades and texture. EasyTrain uses the color image segmentation algorithm developed by James Bruce, Tucker Balch, and Manuela Veloso, Fast and inexpensive color image segmentation for interactive robots. In Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '00), volume 3, pp. 2061-2066, October 2000.

Several pictures of the field and the objects on the field were taken by using the Tekkotsu Raw Camera Viewer of the ERS-7 robot in the ControllerGUI tool. However, before any pictures are taken, the camera has to get the full resolution of the image. In order to set up the camera, the parameter "Take Snapshots" must be created in the dialog box of the ControllerGUI tool. Then in the command box of this parameter, the following commands must be added, saved, and selected:

```
!set vision.rawcam_interval=1000
!set vision.rawcam_transport=tcp
!set vision.rawcam_y_skip=1
!set vision.rawcam_uv_skip=1
!set vision.rawcam_compression=none
```

The raw camera is converted to view it in its YUV format instead of its RGB format in order to distinguish the differences in the intensity of the objects on the field. The pictures can also be taken using a camera demo behavior for conveniance. Using a camera behavior, he user can point the AIBO's camera towards the desired image and tap its head to capture the image. The information in "Take Snapshots" parameter along with the gain and the shutter speed can be adjusted in the Tekkotsu configuration file due to the AIBO taking pictures without the use of ControllerGUI. The camera behavior is as follows:

Files Used

A. SC_CameraBehavior.h
    - Defines the purpose of each variable and event. Creates the event that allows the camera to take a picture what the head button of the robot is pressed once.
B. SC_CameraBehavior.cc
    - Implements the use of the camera by pressing the head button of the robot once. Allows the format and resolution of the picture to be adjusted.

The vision segmentation is only defined for the colors that were significant to the robot during the competition: blue, yellow, orange, and green. Any area of color that is not defined in EasyTrain appears as empty gray space and is disregarded by the robot.

The vision must be recalibrated for every change in environment due to change of lighting. Although to humans, lighting in different rooms do not always seem to change dramatically, the ERS-7 robot will see the color images differently even when a minuscule change in lighting occurs. The Raw Camera Viewer in the ControllerGUI tool allows the user to change the contrast of the camera by adjusting the gain and the shutter speed of the camera.

The gain of the camera has three possible parameters: high, mid, and low. When the gain is adjusted, it will brighten or darken an image. However, the higher the gain of the camera is adjusted, the higher the noise level in the image is increased. The noise level has to be considered because it will cause static in the image. The gain can be adjusted in the ControllerGUI tool by typing the following script into the "Send Input" window:

```
!set vision.gain=[low/mid/high]
```

The shutter speed of the camera has three possible parameters: fast, mid, and slow. When the shutter speed is adjusted, it will also brighten or darken an image. However, the faster the shutter speed of the camera, the more motion blur is increased. The motion blur has to be considered because it will cause some images to blend together because the intensity of the image will be lowered. The shutter speed can be adjusted in the ControllerGUI tool by typing the following script into the "Send Input" window:

```
!set vision.shutter_speed=[slow/mid/fast]
```

Once pictures are taken, the images collected can be opened up in EasyTrain. In the "controls" bar of EasyTrain, the colors desired for calibration can be named. The "Color Spectrum" window provides a choice of color pixels that were captured in the image. Different areas of the image will correspond with various pixels in the spectrum. The colors are defined by selecting the corresponding pixels in the spectrum. The RGB image viewer allows the user to see the image in its real full color form. The Segmented Viewer allows the user to see the image in its segmented color form displaying only the colors defined by the user.

After the segmentation is finished and saved, EasyTrain will create three versions of the file: the color file (.col), the threshold file (.tm), and the spectrum file (.spec). The files are to be saved in the configuration folder in the Tekkotsu project folder. The configuration for Tekkotsu, the "tekkotsu.cfg" file, must be altered to correspond with the new color calibration by calling the color and threshold files.

Vision calibration must be the first task to complete before moving on because the AIBO is programmed to perform actions based on its vision. The EasyTrain vision tool makes vision calibration easy. The actual process of creating the calibration is quick, leaving more time for other tasks. Although the EasyTrain vision tool makes calibration simple, it

is very limiting. The AIBO will only recognize objects as color rather than using the shape or appearance of the objects. For example, the dog may be commanded to chase a pink ball, but if it recognizes anything else in the room as pink, the AIBO will confuse the other pink items as the ball, and chase the other pink items as a result. Different objects in a room may carry pigments of the defined colors in the calibration; therefore, its surrounding environment can easily distract the AIBO. Lighting can dramatically affect the AIBO's vision as well. A bright light can cause an item to reflect a bright white or light yellow color, causing the object to disappear from the AIBO's sight.

VII. Results

Using the described strategy, the SpelBots successfully qualified and competed in the 2006 RoboCup World Championship in Bremen, Germany during the summer of 2006. Although the goal of qualifying and competing was reached, the SpelBots were not able to secure a win with our three matches.

	First Round		Intermediate Round
	UChile1 *(Department of Electrical Engineering, Universidad de Chile)*	Wright Eagle *(University of Science and Technology of China)*	Northern Bites *(Bowdoin College Brunswick, Maine)*
SpelBots *(Spelman College Atlanta, Georgia)*	0:4	1:5	0:1

Our team also gained the ultimate learning tool of experience during the competitions. Other teams and researchers provided useful insights and again welcomed the Spelman College's robotics team into the world of RoboCup. The team looks forward to many years of success in RoboCup and other applications of intelligent agents and robotics.

Conclusion

The SpelBots team will continue to study and learn improved techniques to incorporate into developing code. The team seeks to utilize a more precise and dynamic vision system. As the vision tools plays the pivotal role in our coding scheme, Any enhancements in the vision will allow the SpelBots to improve their localization and strategy.

Acknowledgments

The SpelBots team along with its advisors Dr. Andrew Williams and Dr. Andrea Lawrence gratefully acknowledge the continuous support given by the General Electric Company, NASA, and the Boeing Company. The SpelBots have also received an overwhelming abundance of support from Dr. Beverly Tatum and the Spelman College family. The SpelBots team thanks the organizers of RoboCup 2006 for a well-managed and executed competition. Additionally, the present members of the SpelBots team acknowledge the past contributions from all of the previous team members.

Source Code

```
#include "GoalieHeadFollowNode.h"
#include "Motion/HeadPointerMC.h"
#include "Motion/WalkMC.h"
#include "Motion/MMAccessor.h"
#include "Events/VisionObjectEvent.h"
#include "Shared/WorldState.h"
#include "Behaviors/Transitions/TimeOutTrans.h"
#include "Behaviors/Transitions/VisualTargetCloseTrans.h"
#include "SC_GoalieVisualTarget.h"

void GoalieHeadFollowNode::DoStart() {
     StateNode::DoStart();

     headpointer_id = motman-
>addPersistentMotion(SharedObject<HeadPointerMC>());

     erouter-
>addListener(this,EventBase::visObjEGID,tracking);
}

void GoalieHeadFollowNode::DoStop() {
     erouter->removeListener(this);

     motman->removeMotion(headpointer_id);
     headpointer_id=MotionManager::invalid_MC_ID;


     StateNode::DoStop();
}

//this could be cleaned up event-wise (only use a timer
when out of view)
void GoalieHeadFollowNode::processEvent(const EventBase&
event) {
     static float horiz=0,vert=0;
     const VisionObjectEvent *ve = dynamic_cast<const
VisionObjectEvent*>(&event);
     if(ve!=NULL &&
event.getTypeID()==EventBase::statusETID) {
          horiz=ve->getCenterX();
          vert=ve->getCenterY();
     } else
          return;

     //cout << "Pos: " << horiz << ' ' << vert << endl;
```

```
     double tilt=state->outputs[HeadOffset+TiltOffset]-
vert*M_PI/6;
     double pan=state->outputs[HeadOffset+PanOffset]-
horiz*M_PI/7.5;
     if(tilt>outputRanges[HeadOffset+TiltOffset][MaxRange])

     tilt=outputRanges[HeadOffset+TiltOffset][MaxRange];
     if(tilt<outputRanges[HeadOffset+TiltOffset][MinRange]*
3/4)

     tilt=outputRanges[HeadOffset+TiltOffset][MinRange]*3/4
;
     if(pan>outputRanges[HeadOffset+PanOffset][MaxRange]*2/
3)

     pan=outputRanges[HeadOffset+PanOffset][MaxRange]*2/3;
     if(pan<outputRanges[HeadOffset+PanOffset][MinRange]*2/
3)

     pan=outputRanges[HeadOffset+PanOffset][MinRange]*2/3;
     {MMAccessor<HeadPointerMC>(headpointer_id)-
>setJoints(tilt,pan,0);} //note use of {}'s to limit scope


}

Transition*
GoalieHeadFollowNode::newDefaultLostTrans(StateNode* dest)
{
     return new
TimeOutTrans(dest,1500,EventBase::visObjEGID,tracking);
}

//Transition*
GoalieHeadFollowNode::newDefaultCloseTrans(StateNode* dest)
{
//   return new VisualTargetCloseTrans(dest,tracking);
//}

Transition*
GoalieHeadFollowNode::newDefaultCloseTrans(StateNode* dest)
{
     return new SC_GoalieVisualTarget(dest,tracking);
}



/*! @file
```

```
* @brief    Implements  a motion that allows the goalie to
track the ball using only its head joints, specifically the
pan joint.
* @author Ebony Smith (Creator)
*
* $Author: Ebony Smith $
* $Name: GoalieHeadFollowNode.cc $
* $Revision: 1 $
* $State: GA $
* $Date: 05/20/2006 $
*/


//-*-c++-*-
#ifndef INCLUDED_GoalieHeadFollowNode_h_
#define INCLUDED_GoalieHeadFollowNode_h_

#include "Behaviors/StateNode.h"
#include "Motion/MotionManager.h"

//! a state node for walking towards a visual target
class GoalieHeadFollowNode : public StateNode {
public:
     //!constructor, pass VisionObjectSourceID_t
     GoalieHeadFollowNode(unsigned int obj)
     :
StateNode("GoalieHeadFollowNode","GoalieWalk"),tracking(obj
),
      headpointer_id(MotionManager::invalid_MC_ID)
{}

     //!constructor, pass instance name and
VisionObjectSourceID_t
     GoalieHeadFollowNode(const std::string& nodename,
unsigned int obj)
     :
StateNode("GoalieHeadFollowNode",nodename),tracking(obj),
     headpointer_id(MotionManager::invalid_MC_ID)
{}

     virtual void DoStart();
     virtual void DoStop();

     static std::string getClassDescription() { return
"moves the head to track an object"; }
     virtual std::string getDescription() const { return
getClassDescription(); }
```

```
     //uses head to watch ball, walks towards it
     virtual void processEvent(const EventBase& event);

     virtual Transition* newDefaultLostTrans(StateNode*
dest);  //!< returns a suggested transition for detecting
"lost" condition, but you don't have to use it
     virtual Transition* newDefaultCloseTrans(StateNode*
dest); //!< returns a suggested transition for detecting
"close to target" condition, but you don't have to use it


protected:
          //!constructor, pass class name, instance name,
and VisionObjectSourceID_t
          GoalieHeadFollowNode(const std::string&
classname, const std::string& nodename, unsigned int obj)
          : StateNode(classname,nodename),tracking(obj),
          headpointer_id(MotionManager::invalid_MC_ID)
     {}

     unsigned int tracking; //!< the object being tracked
     MotionManager::MC_ID walker_id; //!< so we can walk
     MotionManager::MC_ID headpointer_id; //!< so we can
point the head at the object

private:
          GoalieHeadFollowNode(const
GoalieHeadFollowNode&); //!< don't call this
     GoalieHeadFollowNode operator=(const
GoalieHeadFollowNode&); //!< don't call this
};


/*! @file
* @brief Implements a motion that allows the goalie to
track the ball using only its head joints, specifically the
pan joint.
* @authorEbony Smith (Creator)
*
* $Author: Ebony Smith $
* $Name: GoalieHeadFollowNode.h $
* $Revision: 1 $
* $State: GA $
* $Date: 05/20/2006 $
*/

#endif
```

```
#include "GoalieWalkNode.h"
#include "Motion/HeadPointerMC.h"
#include "Motion/WalkMC.h"
#include "Motion/MMAccessor.h"
#include "Events/VisionObjectEvent.h"
#include "Shared/WorldState.h"
#include "Behaviors/Transitions/TimeOutTrans.h"
#include "Behaviors/Transitions/VisualTargetCloseTrans.h"

void GoalieWalkNode::DoStart() {
     StateNode::DoStart();

     headpointer_id = motman-
>addPersistentMotion(SharedObject<HeadPointerMC>());
     walker_id = motman-
>addPersistentMotion(SharedObject<WalkMC>());

     erouter-
>addListener(this,EventBase::visObjEGID,tracking);
}

void GoalieWalkNode::DoStop() {
     erouter->removeListener(this);

     motman->removeMotion(headpointer_id);
     headpointer_id=MotionManager::invalid_MC_ID;
     motman->removeMotion(walker_id);
     walker_id=MotionManager::invalid_MC_ID;

     StateNode::DoStop();
}

//this could be cleaned up event-wise (only use a timer
when out of view)
void GoalieWalkNode::processEvent(const EventBase& event) {
     static float horiz=0,vert=0;
     const VisionObjectEvent *ve = dynamic_cast<const
VisionObjectEvent*>(&event);
     if(ve!=NULL &&
event.getTypeID()==EventBase::statusETID) {
          horiz=ve->getCenterX();
          vert=ve->getCenterY();
     } else
          return;

     //cout << "Pos: " << horiz << ' ' << vert << endl;
```

```
     double tilt=state->outputs[HeadOffset+TiltOffset]-
vert*M_PI/6;
     double pan=state->outputs[HeadOffset+PanOffset]-
horiz*M_PI/7.5;
     if(tilt>outputRanges[HeadOffset+TiltOffset][MaxRange])

     tilt=outputRanges[HeadOffset+TiltOffset][MaxRange];
     if(tilt<outputRanges[HeadOffset+TiltOffset][MinRange]*
3/4)

     tilt=outputRanges[HeadOffset+TiltOffset][MinRange]*3/4
;
     if(pan>outputRanges[HeadOffset+PanOffset][MaxRange]*2/
3)

     pan=outputRanges[HeadOffset+PanOffset][MaxRange]*2/3;
     if(pan<outputRanges[HeadOffset+PanOffset][MinRange]*2/
3)

     pan=outputRanges[HeadOffset+PanOffset][MinRange]*2/3;
     {MMAccessor<HeadPointerMC>(headpointer_id)-
>setJoints(tilt,pan,0);} //note use of {}'s to limit scope

     {
          MMAccessor<WalkMC> walker(walker_id);
          if(pan<-.05 || pan>.05)
               walker->setTargetVelocity(0,pan*100,0);
          else
               walker->setTargetVelocity(0,0,0);
     }
}

Transition* GoalieWalkNode::newDefaultLostTrans(StateNode*
dest) {
     return new
TimeOutTrans(dest,1500,EventBase::visObjEGID,tracking);
}

Transition* GoalieWalkNode::newDefaultCloseTrans(StateNode*
dest) {
     return new VisualTargetCloseTrans(dest,tracking);
}



/*! @file
* @brief  Implements a walk that follows the ball
horizontally within the goalie box.
```

```
* @author   Ebony Smith (Creator)
*
* $Author: Ebony Smith $
* $Name: GoalieWalkNode.cc $
* $Revision: 1 $
* $State: GA $
* $Date: 04/2006 $
*/


//-*-c++-*-
#ifndef INCLUDED_GoalieWalkNode_h_
#define INCLUDED_GoalieWalkNode_h_

#include "Behaviors/StateNode.h"
#include "Motion/MotionManager.h"

//! a state node for walking towards a visual target
class GoalieWalkNode : public StateNode {
public:
     //!constructor, pass VisionObjectSourceID_t
     GoalieWalkNode(unsigned int obj)
          :
StateNode("GoalieWalkNode","GoalieWalk"),tracking(obj),
               walker_id(MotionManager::invalid_MC_ID),
headpointer_id(MotionManager::invalid_MC_ID)
     {}

     //!constructor, pass instance name and
VisionObjectSourceID_t
     GoalieWalkNode(const std::string& nodename, unsigned
int obj)
          :
StateNode("GoalieWalkNode",nodename),tracking(obj),
               walker_id(MotionManager::invalid_MC_ID),
headpointer_id(MotionManager::invalid_MC_ID)
     {}

     virtual void DoStart();
     virtual void DoStop();

     static std::string getClassDescription() { return
"walks towards a visual target, using some basic logic for
moving the head to track it"; }
     virtual std::string getDescription() const { return
getClassDescription(); }

     //uses head to watch ball, walks towards it
```

```
    virtual void processEvent(const EventBase& event);

    virtual Transition* newDefaultLostTrans(StateNode*
dest);  //!< returns a suggested transition for detecting
"lost" condition, but you don't have to use it
    virtual Transition* newDefaultCloseTrans(StateNode*
dest); //!< returns a suggested transition for detecting
"close to target" condition, but you don't have to use it


protected:
    //!constructor, pass class name, instance name, and
VisionObjectSourceID_t
    GoalieWalkNode(const std::string& classname, const
std::string& nodename, unsigned int obj)
          : StateNode(classname,nodename),tracking(obj),
               walker_id(MotionManager::invalid_MC_ID),
headpointer_id(MotionManager::invalid_MC_ID)
    {}

    unsigned int tracking; //!< the object being tracked
    MotionManager::MC_ID walker_id; //!< so we can walk
    MotionManager::MC_ID headpointer_id; //!< so we can
point the head at the object

private:
    GoalieWalkNode(const GoalieWalkNode&); //!< don't call
this
    GoalieWalkNode operator=(const GoalieWalkNode&); //!<
don't call this
};

/*! @file
* @brief    Implements a walk that follows the ball
horizontally within the goalie box.
* @author Ebony Smith (Creator)
*
* $Author: Ebony Smith $
* $Name: GoalieWalkNode.h $
* $Revision: 1 $
* $State: GA $
* $Date: 04/2006 $
*/


#endif

#ifndef INCLUDED_SC_Attacker2_h_
```

```
#define INCLUDED_SC_Attacker2_h_

#include "Behaviors/Transition.h"
#include "Behaviors/Nodes/WalkToTargetNode.h"
#include "Behaviors/Nodes/WalkNode.h"
#include "Behaviors/Demos/ExploreMachine.h"
#include "Behaviors/Transitions/TimeOutTrans.h"
#include "Behaviors/Transitions/VisualTargetTrans.h"
#include "Behaviors/Nodes/OutputNode.h"
#include "Behaviors/Nodes/MotionSequenceNode.h"
#include "Behaviors/Nodes/GroupNode.h"
#include "Sound/SoundManager.h"
#include "Shared/ProjectInterface.h"
#include "Behaviors/Transitions/VisualTargetCloseTrans.h"
//#include "Behaviors/Nodes/KickNode.h"
#include "Behaviors/Nodes/HeadPointerNode.h"
#include "Behaviors/Transitions/CompletionTrans.h"
#include "Behaviors/StateNode.h"
#include "VisionHeader.h"
#include "GoalieWalkNode.h"


class SC_Attacker2 : public StateNode {
protected:
     StateNode *SC_Start_Node;

public:
     SC_Attacker2() : StateNode("SC_Attacker2"),
SC_Start_Node(NULL) {}

     void SC_Attacker2::DoStart() {
          StateNode::DoStart();
          SC_Start_Node->DoStart();
     }

     virtual void setup() {
          StateNode::setup();




          GroupNode * SC_Start_Walk_And_Pan = new
GroupNode(getName()+"::SC_Start_Walk_And_Pan");
          addNode(SC_Start_Walk_And_Pan);
          {
               WalkNode * SC_Walk_Forward_Node = new
WalkNode(SC_Start_Walk_And_Pan->getName()+"::TurnInPlace",
150 , 0 , 0);
```

```
                SC_Walk_Forward_Node->setVelocity(150,0,0);
                SC_Start_Walk_And_Pan-
>addNode(SC_Walk_Forward_Node);


                LargeMotionSequenceNode * SC_Pan_Head_Node =
new LargeMotionSequenceNode(SC_Start_Walk_And_Pan-
>getName()+"::PanHead","/ms/data/motion/begnattk.mot",true)
;
                SC_Start_Walk_And_Pan-
>addNode(SC_Pan_Head_Node);
          }

          GroupNode * SC_Attacker2_Kick_Node = new
GroupNode(getName()+"::SC_Attacker2_Kick_Node");
          addNode(SC_Attacker2_Kick_Node);
          {

                LargeMotionSequenceNode * SC_Kick_Node = new
LargeMotionSequenceNode(SC_Attacker2_Kick_Node-
>getName()+"::KickBall","SC_Shkic.mot");
                SC_Attacker2_Kick_Node-
>addNode(SC_Kick_Node);

          }

          WalkToTargetNode * SC_Chase_Node = new
WalkToTargetNode(visOrangeSID);
          SC_Chase_Node->setName(getName()+"::Chase");
          addNode(SC_Chase_Node);

          GroupNode * SC_Turn_Straight_Head_Node = new
GroupNode(getName()+"::SC_Turn_Straight_Head_Node");
          addNode(SC_Turn_Straight_Head_Node);
          {
                WalkNode * SC_Turn_360 = new
WalkNode(SC_Turn_Straight_Head_Node-
>getName()+"::SC_Turn_360", 0 , 0 , .85);
                SC_Turn_360->setVelocity(0,0,.85);
                SC_Turn_Straight_Head_Node-
>addNode(SC_Turn_360);


                MediumMotionSequenceNode * panhead = new
MediumMotionSequenceNode(getName()+"::PanHead","/ms/data/mo
tion/turnpan.mot",true);
                  SC_Turn_Straight_Head_Node-
>addNode(panhead);
```

```
          }


          GroupNode * SC2_Turn_Straight_Head_Node = new
GroupNode(getName()+"::SC2_Turn_Straight_Head_Node");
          addNode(SC2_Turn_Straight_Head_Node);
          {
               WalkNode * SC_Turn_360 = new
WalkNode(SC2_Turn_Straight_Head_Node-
>getName()+"::SC_Turn_360", 0 , 0 , .85);
               SC_Turn_360->setVelocity(0,0,.85);
               SC2_Turn_Straight_Head_Node-
>addNode(SC_Turn_360);

               SmallMotionSequenceNode *
SC_Straight_Head_Node = new
SmallMotionSequenceNode(SC2_Turn_Straight_Head_Node-
>getName()+"::PanHead","/ms/data/motion/str8head.pos",true)
;
               SC2_Turn_Straight_Head_Node-
>addNode(SC_Straight_Head_Node);

          }


          GroupNode * SC_Walk_And_Pan = new
GroupNode(getName()+"::SC_Walk_And_Pan");
          addNode(SC_Walk_And_Pan);
          {
               WalkNode * SC_Forward_Node = new
WalkNode(SC_Walk_And_Pan->getName()+"::SC_Forward_Node",
150 , 0 , 0);
               SC_Forward_Node->setVelocity(150,0,0);
               SC_Walk_And_Pan->addNode(SC_Forward_Node);


               LargeMotionSequenceNode * SC_Pan_Node = new
LargeMotionSequenceNode(SC_Walk_And_Pan-
>getName()+"::SC_Pan_Node","/ms/data/motion/newpan.mot",tru
e);
               SC_Walk_And_Pan->addNode(SC_Pan_Node);
          }



          WalkNode * SC_360 = new
WalkNode(getName()+"::SC_360", 0 , 0 , .85);
          SC_360->setVelocity(0,0,.85);
```

```
          addNode(SC_360);


          WalkNode * SC2_360 = new
WalkNode(getName()+"::SC2_360", 0 , 0 , .85);
          SC2_360->setVelocity(0,0,.85);
          addNode(SC2_360);


          SmallMotionSequenceNode * SC_Straight_Node = new
SmallMotionSequenceNode(getName()+"::PanHead","/ms/data/mot
ion/atkkpana.pos",true);
          addNode(SC_Straight_Node);

          HeadPointerNode * frontNode = new
HeadPointerNode("UpNode");
          addNode(frontNode);
          frontNode-> getMC()->lookAtPoint(200,0,100,30);


          HeadPointerNode * downNode = new
HeadPointerNode("downNode");
          addNode(downNode);
          downNode-> getMC()->lookAtPoint(200,0,30,30);


          GroupNode * SC_Pan_Pounce_2_Node = new
GroupNode(getName()+"::SC_Pan_Pounce_2_Node");
          addNode(SC_Pan_Pounce_2_Node);
          {
               LargeMotionSequenceNode * SC_Pan_Only_Node =
new LargeMotionSequenceNode(SC_Pan_Pounce_2_Node-
>getName()+"::PanHead","/ms/data/motion/newpan.mot",true);
               SC_Pan_Pounce_2_Node-
>addNode(SC_Pan_Only_Node);

               StateNode *SC_Pounce_Node= new
MediumMotionSequenceNode(SC_Pan_Pounce_2_Node-
>getName()+"::stand","/ms/data/motion/pounce.pos",false);
               SC_Pan_Pounce_2_Node-
>addNode(SC_Pounce_Node);

          }
          StateNode *SC_Test_Node= new
MediumMotionSequenceNode(getName()+"::stand","/ms/data/moti
on/pounce.pos",false);
          addNode(SC_Test_Node);
```

```
          Transition * tmptrans=NULL;
          Transition * kicktrans=NULL;
          //Transition * ctrans=NULL;


          //starts out exploring
          SC_Start_Node=SC_Start_Walk_And_Pan;
          SC_Start_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Chase_Node,visOrangeSID));
          SC_Start_Walk_And_Pan->addTransition(new
VisualTargetCloseTrans(SC_Attacker2_Kick_Node,visOrangeSID,
160));
          SC_Start_Walk_And_Pan->addTransition(new
TimeOutTrans(frontNode,3500));
          //SC_Start_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));
          //SC_Start_Walk_And_Pan->addTransition(new
VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID, 160));


          //SC_Chase_Node Transistions
          SC_Chase_Node->addTransition(new
VisualTargetCloseTrans(SC_Attacker2_Kick_Node,visOrangeSID,
160));
          SC_Chase_Node-
>addTransition(tmptrans=SC_Chase_Node-
>newDefaultLostTrans(SC_Pan_Pounce_2_Node));
          //SC_Chase_Node->addTransition(new
VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID, 160));


          //SC_Attacker2_Kick_Node Transistions
          SC_Attacker2_Kick_Node->addTransition(new
TimeOutTrans(SC_Pan_Pounce_2_Node,1000));

          //SC_Pan_Pounce_2_Node Transistions
          SC_Pan_Pounce_2_Node->addTransition(new
TimeOutTrans(frontNode,2500));
          SC_Pan_Pounce_2_Node->addTransition(new
VisualTargetTrans(SC_Chase_Node,visOrangeSID));
          SC_Pan_Pounce_2_Node->addTransition(new
VisualTargetCloseTrans(SC_Attacker2_Kick_Node,visOrangeSID,
160));
          //SC_Pan_Pounce_2_Node->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));
```

```
          //SC_Pan_Pounce_2_Node->addTransition(new
VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID, 160));

          frontNode->addTransition(new
TimeOutTrans(SC_360,500));

          //SC_Turn_360 Transitions
          SC_360->addTransition(new
VisualTargetCloseTrans(SC_Attacker2_Kick_Node,visOrangeSID,
160));
          SC_360->addTransition(new
VisualTargetTrans(SC_Chase_Node,visOrangeSID));
          SC_360->addTransition(new
TimeOutTrans(downNode,6000));
          //SC_360->addTransition(new
VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID, 160));
          //SC_360->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));


          downNode->addTransition(new
TimeOutTrans(SC_Turn_Straight_Head_Node,500));

          SC_Turn_Straight_Head_Node->addTransition(new
VisualTargetCloseTrans(SC_Attacker2_Kick_Node,visOrangeSID,
160));
          SC_Turn_Straight_Head_Node->addTransition(new
VisualTargetTrans(SC_Chase_Node,visOrangeSID));
          SC_Turn_Straight_Head_Node->addTransition(new
TimeOutTrans(SC_Walk_And_Pan,6000));
          //SC_Turn_Straight_Head_Node->addTransition(new
VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID, 160));
          //SC_Turn_Straight_Head_Node->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));


          SC_Walk_And_Pan->addTransition(new
VisualTargetCloseTrans(SC_Attacker2_Kick_Node,visOrangeSID,
160));
          SC_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Chase_Node,visOrangeSID));
          SC_Walk_And_Pan->addTransition(new
TimeOutTrans(frontNode,3000));
          //SC_Walk_And_Pan->addTransition(new
VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID, 160));
          //SC_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));
```

```
          //preload the sounds so we don't pause on
tranisitions
          sndman->LoadFile("cutey.wav");
          sndman->LoadFile("barkmed.wav");
          sndman->LoadFile("whimper.wav");
          sndman->LoadFile("fart.wav");
     }



     void SC_Attacker2::teardown() {
          //release the sounds
          sndman->ReleaseFile("cutey.wav");
          sndman->ReleaseFile("barkmed.wav");
          sndman->ReleaseFile("whimper.wav");
          sndman->ReleaseFile("fart.wav");
          StateNode::teardown();
     }

private:
     SC_Attacker2(const SC_Attacker2&);
     //!< don't call;just satisfies the compiler
     SC_Attacker2 operator=(const SC_Attacker2&);      //!<
don't call;just satisfies the compiler

};

#endif

// Author    : Andrea Roberson
// Date      : 06/2006
// Description:     Main attacker StateNode that implements
attacker state machine



#include "SC_CameraBehavior.h"
#include "Events/EventRouter.h"
#include "Events/TextMsgEvent.h"
#include "Shared/ERS210Info.h"
#include "Shared/ERS220Info.h"
#include "Shared/ERS7Info.h"
#include "Wireless/Socket.h"
#include "Shared/WorldState.h"
#include "Sound/SoundManager.h"
```

```
#include "Shared/Config.h"
#include "Shared/ProjectInterface.h"
#include "Motion/LedMC.h"
#include "Motion/MMAccessor.h"

#include "Vision/FilterBankGenerator.h"
#include "Vision/RawCameraGenerator.h"
#include "Vision/InterleavedYUVGenerator.h"
#include "Vision/JPEGGenerator.h"

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>

void
SC_CameraBehavior::DoStart() {
     BehaviorBase::DoStart();
     if(state->robotDesign&WorldState::ERS210Mask) {

     camera_click.setSourceID(ERS210Info::HeadFrButOffset);
     } else if(state->robotDesign&WorldState::ERS220Mask) {

     camera_click.setSourceID(ERS220Info::HeadFrButOffset);
     } else if(state->robotDesign&WorldState::ERS7Mask) {

     camera_click.setSourceID(ERS7Info::HeadButOffset);
     }

     initIndex();
     sndman->LoadFile("camera.wav");
     ledID=motman-
>addPersistentMotion(SharedObject<LedMC>());
     erouter->addListener(this,camera_click);
     erouter->addListener(this,EventBase::textmsgEGID);
}

void SC_CameraBehavior::DoStop() {
     erouter->removeListener(this);
     sndman->ReleaseFile("camera.wav");
     motman->removeMotion(ledID);
     BehaviorBase::DoStop();
}


/*! The format used depends on the current config settings.
If JPEG
```

```
 *  is the current choice, then a JPEG file will be
written.
 *  Otherwise, RawCameraGenerator::SaveFile() will be
called.
 */
void
SC_CameraBehavior::processEvent(const EventBase& e) {
     if(e.getGeneratorID()==EventBase::textmsgEGID) {
          const TextMsgEvent * txt=dynamic_cast<const
TextMsgEvent*>(&e);
          if(txt==NULL || txt->getText()!="camera")
               return;
     } else if(e.shorterThan(camera_click))
          return;

     {
          MMAccessor<LedMC> leds(ledID);
          leds->cset(FaceLEDMask,2.0/3.0);
          leds->set(TopBrLEDMask,1);
     }


     if(config-
>vision.rawcam_compression==Config::vision_config::COMPRESS
_NONE) {
          //this is our own odd little format, would be
nice to save a TIFF or something instead

          // open file
          //FILE * f=openNextFile(".raw");  *** ORIG
          FILE * f=openNextFile(".raw");
          if(f==NULL) //error message already displayed in
openNextFile()
               return;

          //! write actual image data
          if(config-
>vision.rawcam_encoding==Config::vision_config::ENCODE_COLO
R) {
               FilterBankGenerator *
gen=ProjectInterface::defInterleavedYUVGenerator; // just
an alias for readability
               gen-
>selectSaveImage(ProjectInterface::doubleLayer,InterleavedY
UVGenerator::CHAN_YUV);
               unsigned int len=gen->SaveFileStream(f);
               if(len==0) {
                    serr->printf("Error saving file\n");
```

```
                    sndman->PlayFile(config-
>controller.error_snd);
                    return;
               }
          } else if(config-
>vision.rawcam_encoding==Config::vision_config::ENCODE_SING
LE_CHANNEL) {
               FilterBankGenerator *
gen=ProjectInterface::defRawCameraGenerator; // just an
alias for readability
               gen-
>selectSaveImage(ProjectInterface::doubleLayer,config-
>vision.rawcam_channel);
               unsigned int len=gen->SaveFileStream(f);
               if(len==0) {
                    serr->printf("Error saving file\n");
                    sndman->PlayFile(config-
>controller.error_snd);
                    return;
               }
          }

          // close file
          fclose(f);

     } else if(config-
>vision.rawcam_compression==Config::vision_config::COMPRESS
_JPEG) {
          //save a JPEG image
          JPEGGenerator * jpeg=NULL; // we'll set this to
pick between the color jpeg or a single channel grayscale
jpeg
          unsigned int chan=0; // and this will hold the
channel to use out of that jpeg generator
          if(config-
>vision.rawcam_encoding==Config::vision_config::ENCODE_COLO
R)

     jpeg=dynamic_cast<JPEGGenerator*>(ProjectInterface::de
fColorJPEGGenerator);
          else if(config-
>vision.rawcam_encoding==Config::vision_config::ENCODE_SING
LE_CHANNEL) {

     jpeg=dynamic_cast<JPEGGenerator*>(ProjectInterface::de
fGrayscaleJPEGGenerator);
               chan=config->vision.rawcam_channel;
          }
```

```
          if(jpeg!=NULL) {
               unsigned int tmp_q=jpeg->getQuality();
//temporary storage so we can reset the default
//             jpeg->setQuality(92);    set the quality to
100 so that there is virtually no compression
               jpeg->setQuality(100);

               // open file
//             FILE * f=openNextFile(".jpg");    ***orig
               FILE * f=openNextFile(".jpg");
               if(f==NULL) //error message already
displayed in openNextFile()
                    return;

               //! write actual image data
               unsigned char * imgbuf=jpeg-
>getImage(ProjectInterface::doubleLayer,chan);
               unsigned int writ=fwrite(imgbuf,jpeg-
>getImageSize(ProjectInterface::doubleLayer,chan),1,f);
               if(writ==0) {
                    serr->printf("Error saving file\n");
                    sndman->PlayFile(config-
>controller.error_snd);
                    return;
               }

               // close file
               fclose(f);

               jpeg->setQuality(tmp_q);
          }
     }


     {
          MMAccessor<LedMC> leds(ledID);
          leds->clear();
          leds->flash(TopBrLEDMask,700);
          leds->flash(TopLLEDMask|TopRLEDMask,500);
          leds->flash(MidLLEDMask|MidRLEDMask,300);
          leds->flash(BotLLEDMask|BotRLEDMask,100);
     }

     sout->printf("done\n");
}

FILE *
SC_CameraBehavior::openNextFile(const std::string& ext) {
```

```
     FILE * f=fopen(getNextName(ext).c_str(),"w+");
     if(f==NULL) {
          serr->printf("Error opening file\n");
          sndman->PlayFile(config->controller.error_snd);
          return NULL;
     }
     sndman->PlayFile("camera.wav");
     return f;
}


std::string
SC_CameraBehavior::getNextName(const std::string& ext) {
     char tmp[100];
     snprintf(tmp,100,"data/img%05d%s",index++,ext.c_str())
;
     std::string ans=config->portPath(tmp);
     sout->printf("Saving `%s'...",ans.c_str());
     return ans;
}

void
SC_CameraBehavior::initIndex() {
     std::string path=config->portPath("data/");
     DIR* dir=opendir(path.c_str());
     if(dir==NULL) {
          serr->printf("bad path: `%s'\n",path.c_str());
          return;
     }
     struct dirent * ent=readdir(dir);
     while(ent!=NULL) {
          struct stat s;
          std::string fullpath=path+ent->d_name;
          int err=stat(fullpath.c_str(),&s);
          if(err!=0) {
               serr->printf("File disappeared:
%s\n",fullpath.c_str());
               return;
          }
          if((s.st_mode&S_IFDIR)==0 && strncasecmp(ent-
>d_name,"IMG",3)==0) {
               unsigned int cur=atoi(&ent->d_name[3]);
               if(cur>index)
                    index=cur;
          }
          ent=readdir(dir);
     }
     closedir(dir);
```

```
    index++; //set index to next unused
    sout->printf("The next saved image will go to
%simg%05d\n",path.c_str(),index);
}



// Author    : Ashley Johnson
// Date      : 06/2006
// Description:    Implements the use of the camera by
pressing the head button of the robot once.
//                           Allows the format and
resolution of the picture to be adjusted.


//-*-c++-*-
#ifndef INCLUDED_SC_CameraBehavior_h_
#define INCLUDED_SC_CameraBehavior_h_

#include "Behaviors/BehaviorBase.h"
#include "Motion/MotionManager.h"
#include "Shared/Config.h"

//! Will take images and write to log file
/*! Press the head button to take a picture, back button to
write to memory
 *  stick.  The leds will flash when finished writing.
 *
 *  The reason for this is to provide sample code for
accessing vision
 *  data, and also simply because we should have a way to
save
 *  pictures to memstick instead of relying solely on
having wireless
 *  to transmit them over.
 *
 *  Image format is chosen by current config settings for
the
 *  Config::vision_config::rawcam_compression and
 *  Config::vision_config::rawcam_channel.  However, the
double
 *  resolution layer is always saved instead of whatever
the current
 *  config skip value indicates.
 */
class SC_CameraBehavior : public BehaviorBase {
 public:
    //! constructor, just sets up the variables
```

```
     SC_CameraBehavior()
          : BehaviorBase("SC_CameraBehavior"),
camera_click(EventBase::buttonEGID,0,EventBase::deactivateE
TID,150), index(0), ledID(MotionManager::invalid_MC_ID)
     {
               /***  Set vision to these parameters to use
Vision train
               !set vision.rawcam_interval=1000
               !set vision.rawcam_transport=tcp
               !set vision.rawcam_y_skip=1
               !set vision.rawcam_uv_skip=1
               !set vision.rawcam_compression=none
               ***/
          /**  config->vision.rawcam_interval=1000;  // ***
Remove these if you want the default vision.rawcam
parameters
               config->vision.rawcam_transport=1; // 1=tcp,
0=udp
               config->vision.rawcam_y_skip=1;
               config->vision.rawcam_uv_skip=1;
               //   config-
>vision.rawcam_compression=Config::vision_config::COMPRESS_
NONE;
               config-
>vision.rawcam_compression=Config::vision_config::COMPRESS_
JPEG;
               config->vision.gain = 3;
               config->vision.shutter_speed = 1;

               ***/

     }

     //! Register for events
     virtual void DoStart();

     //! Removes its two motion commands
     virtual void DoStop();

     //! Handles event processing - determines which
generator to save from and writes to current file
     virtual void processEvent(const EventBase& e);

     static std::string getClassDescription() { return
"Push head button to save a picture"; }
     virtual std::string getDescription() const { return
getClassDescription(); }
```

```
 protected:
     //! opens the next file to be saved to (with @a ext
extension on the file name)
     FILE * openNextFile(const std::string& ext);

     //! returns the path and name of the next file to be
saved to (with @a ext extension on the file name)
     std::string getNextName(const std::string& ext);

     //! scans the /ms/data directory for image files and
assigns the next unused index to #index
     void initIndex();

     EventBase camera_click; //!< event mask for taking a
picture (head button)
     unsigned int index; //!< the index to use for the next
image saved

     MotionManager::MC_ID ledID; //!< the id of the LedMC
used to signal completion
};


// Author    : Ashley Johnson
// Date      : 06/2006
// Description:      Defines the purpose of each variable
and event.
//                              Creates the event that
allows the camera to take a picture what the head button of
the robot is pressed once.



#endif

#ifndef _SC_FindOrangeBallEvent_h_
#define _SC_FindOrangeBallEvent_h_

#include "DualCoding/DualCoding.h"
#include "Vision/RegionGenerator.h"
#include "Events/VisionObjectEvent.h"
#include "VisionHeader.h"

using namespace DualCoding;

class SC_FindOrangeBallEvent : public
VisualRoutinesBehavior {
public:
```

```
     SC_FindOrangeBallEvent() :
VisualRoutinesBehavior("SC_FindOrangeBallEvent") {}

     void DoStart() {
          VisualRoutinesBehavior::DoStart();
          erouter->addListener(this,
EventBase::visRegionEGID);
          found=0;

     } // end DoStart                        taking this out
because I want to process only ONE camera frame.  Otherwise
many visRegion events are generated and the max Area's are
replicated and too big

     void processEvent(const EventBase &event) {

          if (event.getGeneratorID() ==
EventBase::visRegionEGID) {
               found = 0; // say the event is not found
every time a visRegion event is posted

               camSkS.clear();
               camShS.clear();

               NEW_SKETCH(camFrame, uchar,
sketchFromSeg());
               NEW_SHAPEVEC(blob_shapes, BlobData,
BlobData::extractBlobs(camFrame,70));  // ball is about
area=50 from center to front of goalie box.

               NEW_SHAPEVEC(orange_blobs, BlobData,
subset(blob_shapes, isColor("orange")));
               NEW_SHAPEVEC(yellow_blobs, BlobData,
subset(blob_shapes, isColor("yellow")));  // don't want to
kick to yellow goal
/*             NEW_SHAPEVEC(green_blobs, BlobData,
subset(blob_shapes, isColor("green")));
               NEW_SHAPEVEC(yellow_blobs, BlobData,
subset(blob_shapes, isColor("yellow")));

               NEW_SHAPEVEC(blue_blobs, BlobData,
subset(blob_shapes, isColor("blue")));
               NEW_SHAPEVEC(pink_blobs, BlobData,
subset(blob_shapes, isColor("pink")));
*/
               float maxArea = 70;
               SHAPEVEC_ITERATE(orange_blobs, BlobData, b1)
{
```

```
                    int count=0;
                    b1Area = b1->getArea();
                    //cout << "orange blob area #" <<
count++ << " is " << b1Area << endl;
                    // Look for the biggest blob and want
the blob to be less than a ball right in front of aibo's
nose, about area=16272
                    if  ((b1Area > maxArea) && (b1Area <
17000)) {
                         Point maxShape = b1-
>getCentroid();
                         //cout << ">>>In
SC_FindOrangeBallEvent process event: Center X is " << b1-
>getCentroid().coordX()
                         //   << " Center Y is " << b1-
>getCentroid().coordY() << " Area is " << b1->getArea() <<
endl;
                         orangeBallCenterX =  b1-
>getCentroid().coordX();
                         orangeBallCenterY =  b1-
>getCentroid().coordY();

                         NEW_SHAPE(goal, PointData, new
PointData(camShS, maxShape));

                         maxArea = b1Area;
                         goal-
>setColor(ProjectInterface::getColorRGB("orange"));
                         goal->setName("orangeBall"); //
this will allow me to refer to this blob as orangeBall in a
VisualRoutineStateNode

                         // Get the height and width of the
orange blobs
                         // determine the height of the
yellow blob
                         float blobHeight = b1-
>bottomLeft.coordY() - b1->topLeft.coordY();

                         // determine the width of the
orange blob
                         float blobWidth = b1-
>topRight.coordX() - b1->topLeft.coordX();

                         // Determine if it is roughly a
square .  That is, height is within 20 percent of the
width)
```

```
                        if ((b1Area <1000) && (blobHeight
< (0.80 * blobWidth)) && (blobHeight > (1.20 * blobWidth)))
{  // try to make sure it's a "square" circle
                              found = 0;
                              Point answer=b1-
>getCentroid();
                              NEW_SHAPE(didNotFindIt,
PointData, new PointData(camShS, answer));
                        } else if ((b1Area < 800 ) && (b1-
>bottomLeft.coordY() < 40)){  // try to make sure it's not
seeing an orange banner in the "sky"
                              found = 0;
                              Point answer=b1-
>getCentroid();
                              NEW_SHAPE(didNotFindIt,
PointData, new PointData(camShS, answer));
                        } else {
                              found =1;
                              Point answer=b1-
>getCentroid();
                              NEW_SHAPE(foundIt, PointData,
new PointData(camShS, answer));
                        }


                        // Get the parameters for a vision
object event.  These are normalized for the camera with the
origin at the center and the max ranges for x and y are (-
1,1).
                        dim =
max(camFrame.width,camFrame.height);
                        cw = camFrame.width/dim;
                        ch = camFrame.height/dim;
                        x1 = 2.0f*b1-
>topLeft.coordX()/camFrame.width - cw;
                        x2 = 2.0f*b1-
>bottomRight.coordX()/camFrame.width - cw;
                        y1 = 2.0f*b1-
>topLeft.coordY()/camFrame.height - ch;
                        y2 = 2.0f*b1-
>bottomRight.coordY()/camFrame.height - ch;
                     }

               } END_ITERATE;

               // If found the orange goal then throw an
event
```

```
                // Announce the goal by posting a
VisionObjectEvent
                if (found == 1) {

                     VisionObjectEvent *obj = new
VisionObjectEvent(orangeBallYesSID,EventBase::activateETID,

                   x1, x2, y1, y2,
b1Area/(camFrame.width*camFrame.height), cw, ch,

                   ProjectInterface::defRegionGenerator-
>getFrameNumber());
                     erouter->postEvent(obj);
                     cout << "Posted a vision object event
for orange ball " << endl;
                     return;

                }
           }
     } // end processEvent

     virtual void DoStop() {
          VisualRoutinesBehavior::DoStop();
          erouter->removeListener(this,
EventBase::visRegionEGID);

     }



     float orangeBallCenterX;
     float orangeBallCenterY;

     int found;
     float b1Area; // blob area

     float dim; // maximum camera dimensions
     float cw;  // camera frame width
     float ch;   // camera frame height
     float x1;  //  top left x coordinate
     float x2;  // bottom right x coordinate
     float y1;  // top left y coordinate
     float y2;  // bottom right y coordinate

};

#endif
```

```
#ifndef _SC_FindOrangeBlueEvent_h_
#define _SC_FindOrangeBlueEvent_h_

#include "DualCoding/DualCoding.h"
#include "Vision/RegionGenerator.h"
#include "Events/VisionObjectEvent.h"
#include "VisionHeader.h"

using namespace DualCoding;

class SC_FindOrangeBlueEvent : public
VisualRoutinesBehavior {
public:
     SC_FindOrangeBlueEvent() :
VisualRoutinesBehavior("SC_FindOrangeBlueEvent") {}

     void DoStart() {
          VisualRoutinesBehavior::DoStart();
          erouter->addListener(this,
EventBase::visRegionEGID);
          found=0;

     } // end DoStart                        taking this out
because I want to process only ONE camera frame.  Otherwise
many visRegion events are generated and the max Area's are
replicated and too big

     void processEvent(const EventBase &event) {

          if (event.getGeneratorID() ==
EventBase::visRegionEGID) {
               foundBlue=0; // say the event is not found
every time a visRegion event is posted
               found=0;

               camSkS.clear();
               camShS.clear();

               NEW_SKETCH(camFrame, uchar,
sketchFromSeg());
               NEW_SHAPEVEC(blob_shapes, BlobData,
BlobData::extractBlobs(camFrame,70));  // ball is about
area=50 from center to front of goalie box.

               NEW_SHAPEVEC(orange_blobs, BlobData,
subset(blob_shapes, isColor("orange")));
```

```
/*              NEW_SHAPEVEC(green_blobs, BlobData,
subset(blob_shapes, isColor("green")));
                NEW_SHAPEVEC(yellow_blobs, BlobData,
subset(blob_shapes, isColor("yellow")));
                NEW_SHAPEVEC(blue_blobs, BlobData,
subset(blob_shapes, isColor("blue")));
                NEW_SHAPEVEC(pink_blobs, BlobData,
subset(blob_shapes, isColor("pink")));
*/

                NEW_SHAPEVEC(blue_blobs, BlobData,
subset(blob_shapes, isColor("blue")));

     //*************************************************
                //  Process blue blobs to look for blue goal

     //*************************************************
                float maxAreaBlue = 100;
                SHAPEVEC_ITERATE(blue_blobs, BlobData,
blueBlob) {
                     blueBlobArea = blueBlob->getArea();

                     // Look for the biggest blob and want
the blob to bigger than a blue marker blob, 800 pixels area
                     if  ((blueBlobArea > maxAreaBlue) &&
(blueBlobArea > 800)) {
                          Point maxShape = blueBlob-
>getCentroid();
                          cout << "Center X is " <<
blueBlob->getCentroid().coordX()
                               << " Center Y is " <<
blueBlob->getCentroid().coordY() << endl;
                          blueGoalCenterX =  blueBlob-
>getCentroid().coordX();
                          blueGoalCenterY =  blueBlob-
>getCentroid().coordY();

                          NEW_SHAPE(goal, PointData, new
PointData(camShS, maxShape));

                          maxAreaBlue = blueBlobArea;
                          foundBlue=1;


                     }

                } END_ITERATE;
```

```
    //*************************************************
              //  Process orange blobs to look for blue
goal

    //*************************************************
              float maxArea = 70;
              SHAPEVEC_ITERATE(orange_blobs, BlobData, b1)
{

                   b1Area = b1->getArea();
                   //cout << "orange blob area #" <<
count++ << " is " << b1Area << endl;
                   // Look for the biggest blob and want
the blob to be less than a ball right in front of aibo's
nose, about area=16272
                   if  ((b1Area > maxArea) && (b1Area <
17000)) {
                        Point maxShape = b1-
>getCentroid();
                        //cout << ">>>In
SC_FindOrangeBallEvent process event: Center X is " << b1-
>getCentroid().coordX()
                        //   << " Center Y is " << b1-
>getCentroid().coordY() << " Area is " << b1->getArea() <<
endl;
                        orangeBallCenterX =  b1-
>getCentroid().coordX();
                        orangeBallCenterY =  b1-
>getCentroid().coordY();

                        NEW_SHAPE(goal, PointData, new
PointData(camShS, maxShape));

                        maxArea = b1Area;
                        goal-
>setColor(ProjectInterface::getColorRGB("orange"));
                        goal->setName("orangeBall"); //
this will allow me to refer to this blob as orangeBall in a
VisualRoutineStateNode

                        // Get the height and width of the
orange blobs
                        // determine the height of the
blue blob
                        float blobHeight = b1-
>bottomLeft.coordY() - b1->topLeft.coordY();
```

```
                        // determine the width of the
orange blob
                        float blobWidth = b1-
>topRight.coordX() - b1->topLeft.coordX();

                        // Determine if it is roughly a
square .  That is, height is within 20 percent of the
width)

                        if ((b1Area <1000) && (blobHeight
< (0.80 * blobWidth)) && (blobHeight > (1.20 * blobWidth)))
{  // try to make sure it's a "square" circle
                             found = 0;
                             //Point answer=b1-
>getCentroid();
                             //NEW_SHAPE(didNotFindIt,
PointData, new PointData(camShS, answer));
                        } else if ((b1Area < 800 ) && (b1-
>bottomLeft.coordY() < 40)){  // try to make sure it's not
seeing an orange banner in the "sky"
                             found = 0;
                             //Point answer=b1-
>getCentroid();
                             //NEW_SHAPE(didNotFindIt,
PointData, new PointData(camShS, answer));
                        } else {
                             found =1;
                             Point answer=b1-
>getCentroid();
                             NEW_SHAPE(foundIt, PointData,
new PointData(camShS, answer));
                        }


                        // If found the orange goal then
throw an event
                        // Announce the goal by posting a
VisionObjectEvent

                        // Get the parameters for a vision
object event.  These are normalized for the camera with the
origin at the center and the max ranges for x and y are (-
1,1).
                        dim =
max(camFrame.width,camFrame.height);
                        cw = camFrame.width/dim;
                        ch = camFrame.height/dim;
```

```
                        x1 = 2.0f*b1-
>topLeft.coordX()/camFrame.width - cw;
                        x2 = 2.0f*b1-
>bottomRight.coordX()/camFrame.width - cw;
                        y1 = 2.0f*b1-
>topLeft.coordY()/camFrame.height - ch;
                        y2 = 2.0f*b1-
>bottomRight.coordY()/camFrame.height - ch;
                    }

               } END_ITERATE;



               if  ((found==1) && (foundBlue==1)){

                    VisionObjectEvent *obj = new
VisionObjectEvent(orangeBlueYesSID,EventBase::activateETID,

                   x1, x2, y1, y2,
b1Area/(camFrame.width*camFrame.height), cw, ch,

                   ProjectInterface::defRegionGenerator-
>getFrameNumber());
                     erouter->postEvent(obj);
                     cout << "Posted a vision object event
for orange ball " << endl;
                     found=0;
                     foundBlue=0;
                     return;

               } else if ((found==1) && (foundBlue !=1)) {
                     VisionObjectEvent *obj = new
VisionObjectEvent(orangeBallYesSID,EventBase::activateETID,

                   x1, x2, y1, y2,
b1Area/(camFrame.width*camFrame.height), cw, ch,

                   ProjectInterface::defRegionGenerator-
>getFrameNumber());
                     erouter->postEvent(obj);
                     cout << "Posted a vision object event
for orange ball " << endl;
                     found=0;
                     foundBlue=0;
                     return;
               } else {
                     found=0;
```

```
                    foundBlue=0;
                    return;  // NEEDED to quite posted
events???
               }


          }
     } // end processEvent

     virtual void DoStop() {
          VisualRoutinesBehavior::DoStop();
          erouter->removeListener(this,
EventBase::visRegionEGID);

     }



     float orangeBallCenterX;
     float orangeBallCenterY;

     float blueGoalCenterX;
     float blueGoalCenterY;

     int found;
     int foundBlue;

     float b1Area; // orange blob area
     float blueBlobArea; // orange blob area

     float dim; // maximum camera dimensions
     float cw;  // camera frame width
     float ch;   // camera frame height
     float x1;  //  top left x coordinate
     float x2;  // bottom right x coordinate
     float y1;  // top left y coordinate
     float y2;  // bottom right y coordinate

};

#endif
#ifndef _SC_FindOrangeYellowEvent_h_
#define _SC_FindOrangeYellowEvent_h_

#include "DualCoding/DualCoding.h"
#include "Vision/RegionGenerator.h"
#include "Events/VisionObjectEvent.h"
#include "VisionHeader.h"
```

```
using namespace DualCoding;

class SC_FindOrangeYellowEvent : public
VisualRoutinesBehavior {
public:
     SC_FindOrangeYellowEvent() :
VisualRoutinesBehavior("SC_FindOrangeYellowEvent") {}

     void DoStart() {
          VisualRoutinesBehavior::DoStart();
          erouter->addListener(this,
EventBase::visRegionEGID);
          found=0;

     } // end DoStart                        taking this out
because I want to process only ONE camera frame.  Otherwise
many visRegion events are generated and the max Area's are
replicated and too big

     void processEvent(const EventBase &event) {

          if (event.getGeneratorID() ==
EventBase::visRegionEGID) {
               foundYellow=0; // say the event is not found
every time a visRegion event is posted
               found=0;

               camSkS.clear();
               camShS.clear();

               NEW_SKETCH(camFrame, uchar,
sketchFromSeg());
               NEW_SHAPEVEC(blob_shapes, BlobData,
BlobData::extractBlobs(camFrame,70));  // ball is about
area=50 from center to front of goalie box.

               NEW_SHAPEVEC(orange_blobs, BlobData,
subset(blob_shapes, isColor("orange")));

/*             NEW_SHAPEVEC(green_blobs, BlobData,
subset(blob_shapes, isColor("green")));
               NEW_SHAPEVEC(yellow_blobs, BlobData,
subset(blob_shapes, isColor("yellow")));
               NEW_SHAPEVEC(blue_blobs, BlobData,
subset(blob_shapes, isColor("blue")));
               NEW_SHAPEVEC(pink_blobs, BlobData,
subset(blob_shapes, isColor("pink")));
```

```
*/

               NEW_SHAPEVEC(yellow_blobs, BlobData,
subset(blob_shapes, isColor("yellow")));

     //*************************************************
               //  Process yellow blobs to look for yellow
goal

     //*************************************************
               float maxAreaYellow = 100;
               SHAPEVEC_ITERATE(yellow_blobs, BlobData,
yellowBlob) {
                    yellowBlobArea = yellowBlob->getArea();

                    // Look for the biggest blob and want
the blob to bigger than a yellow marker blob, 800 pixels
area
                    if  ((yellowBlobArea > maxAreaYellow)
&& (yellowBlobArea > 800)) {
                         Point maxShape = yellowBlob-
>getCentroid();
                         cout << "Center X is " <<
yellowBlob->getCentroid().coordX()
                              << " Center Y is " <<
yellowBlob->getCentroid().coordY() << endl;
                         yellowGoalCenterX =  yellowBlob-
>getCentroid().coordX();
                         yellowGoalCenterY =  yellowBlob-
>getCentroid().coordY();

                         NEW_SHAPE(goal, PointData, new
PointData(camShS, maxShape));

                         maxAreaYellow = yellowBlobArea;
                         foundYellow=1;


                    }

               } END_ITERATE;


     //*************************************************
               //  Process orange blobs to look for yellow
goal

     //*************************************************
```

```
                float maxArea = 70;
                SHAPEVEC_ITERATE(orange_blobs, BlobData, b1)
{

                     b1Area = b1->getArea();
                     //cout << "orange blob area #" <<
count++ << " is " << b1Area << endl;
                     // Look for the biggest blob and want
the blob to be less than a ball right in front of aibo's
nose, about area=16272
                     if  ((b1Area > maxArea) && (b1Area <
17000)) {
                          Point maxShape = b1-
>getCentroid();
                          //cout << ">>>In
SC_FindOrangeBallEvent process event: Center X is " << b1-
>getCentroid().coordX()
                          //   << " Center Y is " << b1-
>getCentroid().coordY() << " Area is " << b1->getArea() <<
endl;
                          orangeBallCenterX =  b1-
>getCentroid().coordX();
                          orangeBallCenterY =  b1-
>getCentroid().coordY();

                          NEW_SHAPE(goal, PointData, new
PointData(camShS, maxShape));

                          maxArea = b1Area;
                          goal-
>setColor(ProjectInterface::getColorRGB("orange"));
                          goal->setName("orangeBall"); //
this will allow me to refer to this blob as orangeBall in a
VisualRoutineStateNode

                          // Get the height and width of the
orange blobs
                          // determine the height of the
yellow blob
                          float blobHeight = b1-
>bottomLeft.coordY() - b1->topLeft.coordY();

                          // determine the width of the
orange blob
                          float blobWidth = b1-
>topRight.coordX() - b1->topLeft.coordX();
```

```
                        // Determine if it is roughly a
square .  That is, height is within 20 percent of the
width)

                        if ((b1Area <1000) && (blobHeight
< (0.80 * blobWidth)) && (blobHeight > (1.20 * blobWidth)))
{  // try to make sure it's a "square" circle
                             found = 0;
                             //Point answer=b1-
>getCentroid();
                             //NEW_SHAPE(didNotFindIt,
PointData, new PointData(camShS, answer));
                        } else if ((b1Area < 800 ) && (b1-
>bottomLeft.coordY() < 40)){  // try to make sure it's not
seeing an orange banner in the "sky"
                             found = 0;
                             //Point answer=b1-
>getCentroid();
                             //NEW_SHAPE(didNotFindIt,
PointData, new PointData(camShS, answer));
                        } else {
                             found =1;
                             Point answer=b1-
>getCentroid();
                             NEW_SHAPE(foundIt, PointData,
new PointData(camShS, answer));
                        }


                        // If found the orange goal then
throw an event
                        // Announce the goal by posting a
VisionObjectEvent

                        // Get the parameters for a vision
object event.  These are normalized for the camera with the
origin at the center and the max ranges for x and y are (-
1,1).
                        dim =
max(camFrame.width,camFrame.height);
                        cw = camFrame.width/dim;
                        ch = camFrame.height/dim;
                        x1 = 2.0f*b1-
>topLeft.coordX()/camFrame.width - cw;
                        x2 = 2.0f*b1-
>bottomRight.coordX()/camFrame.width - cw;
                        y1 = 2.0f*b1-
>topLeft.coordY()/camFrame.height - ch;
```

```
                        y2 = 2.0f*b1-
>bottomRight.coordY()/camFrame.height - ch;
                    }

               } END_ITERATE;



               if  ((found==1) && (foundYellow==1)){

                    VisionObjectEvent *obj = new
VisionObjectEvent(orangeYellowYesSID,EventBase::activateETI
D,

                  x1, x2, y1, y2,
b1Area/(camFrame.width*camFrame.height), cw, ch,

                  ProjectInterface::defRegionGenerator-
>getFrameNumber());
                    erouter->postEvent(obj);
                    cout << "Posted a vision object event
for orange ball " << endl;
                    found=0;
                    foundYellow=0;
                    return;

               } else if ((found==1) && (foundYellow !=1))
{
                    VisionObjectEvent *obj = new
VisionObjectEvent(orangeBallYesSID,EventBase::activateETID,

                  x1, x2, y1, y2,
b1Area/(camFrame.width*camFrame.height), cw, ch,

                  ProjectInterface::defRegionGenerator-
>getFrameNumber());
                    erouter->postEvent(obj);
                    cout << "Posted a vision object event
for orange ball " << endl;
                    found=0;
                    foundYellow=0;
                    return;
               } else {
                    found=0;
                    foundYellow=0;
                    return;  // NEEDED to quite posted
events???
               }
```

```
          }
     } // end processEvent

     virtual void DoStop() {
          VisualRoutinesBehavior::DoStop();
          erouter->removeListener(this,
EventBase::visRegionEGID);

     }



     float orangeBallCenterX;
     float orangeBallCenterY;

     float yellowGoalCenterX;
     float yellowGoalCenterY;

     int found;
     int foundYellow;

     float b1Area; // orange blob area
     float yellowBlobArea; // orange blob area

     float dim; // maximum camera dimensions
     float cw;  // camera frame width
     float ch;   // camera frame height
     float x1;  //  top left x coordinate
     float x2;  // bottom right x coordinate
     float y1;  // top left y coordinate
     float y2;  // bottom right y coordinate

};

#endif
#ifndef INCLUDED_SC_Goalie_Lil_Shuff_h_
#define INCLUDED_SC_Goalie_Lil_Shuff_h_

#include "Behaviors/Transition.h"
#include "Behaviors/Nodes/WalkToTargetNode.h"
#include "Behaviors/Nodes/WalkNode.h"
#include "Behaviors/Demos/ExploreMachine.h"
#include "Behaviors/Transitions/TimeOutTrans.h"
#include "Behaviors/Transitions/VisualTargetTrans.h"
#include "Behaviors/Nodes/OutputNode.h"
#include "Behaviors/Nodes/MotionSequenceNode.h"
```

```
#include "Behaviors/Nodes/GroupNode.h"
#include "Sound/SoundManager.h"
#include "Shared/ProjectInterface.h"
#include "Behaviors/Transitions/VisualTargetCloseTrans.h"
#include "Behaviors/Nodes/KickNode.h"
#include "Behaviors/Nodes/HeadPointerNode.h"
#include "Behaviors/Transitions/CompletionTrans.h"
#include "Behaviors/StateNode.h"
#include "VisionHeader.h"
#include "GoalieWalkNode.h"
#include "GoalieHeadFollowNode.h"
#include "SC_GoalieVisualTarget.h"
#include "SC_GoalieShuffTrans.h"

class SC_Goalie_Lil_Shuff : public StateNode {
protected:
     StateNode *SC_Start_Node;

public:
     SC_Goalie_Lil_Shuff() :
StateNode("SC_Goalie_Lil_Shuff"), SC_Start_Node(NULL) {}

     void SC_Goalie_Lil_Shuff::DoStart() {
          StateNode::DoStart();
          SC_Start_Node->DoStart();
     }

     virtual void setup() {
          StateNode::setup();


          LargeMotionSequenceNode * SC_Pan_Head_Node = new
LargeMotionSequenceNode(SC_Pan_Head_Node-
>getName()+"::PanHead","/ms/data/motion/newpan.mot",true);
          addNode(SC_Pan_Head_Node);

          MediumMotionSequenceNode * SC_AK_Pan_Node = new
MediumMotionSequenceNode(getName()+"::PanHead","/ms/data/mo
tion/newpan.mot",true);
          addNode(SC_AK_Pan_Node);

          GoalieWalkNode * SC_Goalie_Shuffle_Node = new
GoalieWalkNode(getName()+"::Chase", visOrangeSID);
          addNode(SC_Goalie_Shuffle_Node);

          GoalieHeadFollowNode * SC_Goalie_Head_Node = new
GoalieHeadFollowNode(getName()+"::SC_Goalie_Head_Node",
visOrangeSID);
```

```
          addNode(SC_Goalie_Head_Node);


          GroupNode * SC_Goalie_Kick_Node = new
GroupNode(getName()+"::SC_Goalie_Kick_Node");
          addNode(SC_Goalie_Kick_Node);
          {
               SmallMotionSequenceNode * kickball = new
SmallMotionSequenceNode(SC_Goalie_Kick_Node-
>getName()+"::KickBall","/ms/data/motion/ekogoal.mot");
               SC_Goalie_Kick_Node->addNode(kickball);
          }


          StateNode *SC_Goalie_Pounce_Node= new
MediumMotionSequenceNode(getName()+"::stand","/ms/data/moti
on/goalsit.pos",false);
          addNode(SC_Goalie_Pounce_Node);

          StateNode *SC_AK_Pounce_Node= new
MediumMotionSequenceNode(getName()+"::stand","/ms/data/moti
on/goalsit.pos",false);
          addNode(SC_AK_Pounce_Node);

          WalkNode * SC_Turn_360_Node = new
WalkNode(getName()+"::SC_Turn_360_Node", 0 , 0 , .85);
          SC_Turn_360_Node->setVelocity(0,0,.85);
          addNode(SC_Turn_360_Node);

          GroupNode * SC_Goalie_Turn_Pan = new
GroupNode(getName()+"::SC_Goalie_Turn_Pan");
          addNode(SC_Goalie_Turn_Pan);
          {
               WalkNode * SC_Turn_Look_Node = new
WalkNode(getName()+"::SC_Turn_Look_Node", 0 , 0 , .85);
               SC_Turn_Look_Node->setVelocity(0,0,.85);
               SC_Goalie_Turn_Pan-
>addNode(SC_Turn_Look_Node);

               MediumMotionSequenceNode * SC_Pan_Look_Node
= new
MediumMotionSequenceNode(getName()+"::PanHead","/ms/data/mo
tion/pan_head.mot",true);
               SC_Goalie_Turn_Pan-
>addNode(SC_Pan_Look_Node);
          }
```

```
          WalkToTargetNode * SC_Back_To_YGoal = new
WalkToTargetNode(yellowGoalSID);
          SC_Back_To_YGoal-
>setName(getName()+"::SC_Back_To_YGoal");
          addNode(SC_Back_To_YGoal);









           /*

           SC_Pan_Head_Node->addTransition(new
TimeOutTrans(SC_Find_YGoal_Node,9000));
           SC_Find_YGoal_Node->addTransition(new
VisualTargetTrans(SC_Back_To_YGoal,yellowGoalSID));
           SC_Back_To_YGoal->addTransition(new
VisualTargetCloseTrans(SC_Find_BGoal_Node,blueGoalSID));
           SC_Find_BGoal_Node->addTransition(new
VisualTargetTrans(SC_Goalie_Pounce_Node,blueGoalSID));

*/









          //starts in a pounce position
          SC_Start_Node=SC_Goalie_Pounce_Node;
          //If the ball is sighted from a distance, shuffle
and follow it horizontally
          //SC_Goalie_Pounce_Node->addTransition(new
VisualTargetTrans(SC_Goalie_Shuffle_Node,visOrangeSID));
          SC_Goalie_Pounce_Node->addTransition(new
SC_GoalieShuffTrans(SC_Goalie_Shuffle_Node,visOrangeSID));
          //If you get close enough to the ball, kick
          //SC_Goalie_Pounce_Node->addTransition(new
VisualTargetCloseTrans(SC_Goalie_Kick_Node,visOrangeSID));
```

```
          SC_Goalie_Pounce_Node->addTransition(new
SC_GoalieVisualTarget(SC_Goalie_Kick_Node,visOrangeSID));
          //After a second, begin panning head in search of
ball
          SC_Goalie_Pounce_Node->addTransition(new
TimeOutTrans(SC_Pan_Head_Node,1500,EventBase::visObjEGID,vi
sOrangeSID));


          //Some definitions for transitions
          Transition * tmptrans=NULL;
          Transition * kicktrans=NULL;

          //While panning head, shuffle and follow it
horizontally
          //SC_Pan_Head_Node->addTransition(tmptrans=new
VisualTargetTrans(SC_Goalie_Shuffle_Node,visOrangeSID));
          SC_Pan_Head_Node->addTransition(tmptrans=new
SC_GoalieShuffTrans(SC_Goalie_Shuffle_Node,visOrangeSID));
          //If you get close enough to the ball, kick
          //SC_Pan_Head_Node->addTransition(tmptrans=new
VisualTargetCloseTrans(SC_Goalie_Kick_Node,visOrangeSID));
          SC_Pan_Head_Node->addTransition(tmptrans=new
SC_GoalieVisualTarget(SC_Goalie_Kick_Node,visOrangeSID));



          //If the ball is lost while shuffling, pan head
in search of it
          SC_Goalie_Shuffle_Node-
>addTransition(tmptrans=SC_Goalie_Shuffle_Node-
>newDefaultLostTrans(SC_Pan_Head_Node));
          //If you get close enough to the ball, kick
          //SC_Goalie_Shuffle_Node->addTransition(kicktrans
=new VisualTargetCloseTrans(SC_Goalie_Kick_Node,
visOrangeSID));
          SC_Goalie_Shuffle_Node->addTransition(kicktrans
=new SC_GoalieVisualTarget(SC_Goalie_Kick_Node,
visOrangeSID));
          SC_Goalie_Shuffle_Node->addTransition(new
TimeOutTrans(SC_Goalie_Head_Node,3000));
          //If you have been shuffling following the ball
for a while, just follow it with your head.
          //SC_Goalie_Shuffle_Node->addTransition(new
TimeOutTrans(SC_Goalie_Head_Node,9000));
```

```
          //If the ball is lost while following with head,
pan head in search of it
          SC_Goalie_Head_Node-
>addTransition(tmptrans=SC_Goalie_Head_Node-
>newDefaultLostTrans(SC_Pan_Head_Node));
          //If you get close enough to the ball, kick
          //SC_Goalie_Head_Node->addTransition(kicktrans
=new VisualTargetCloseTrans(SC_Goalie_Kick_Node,
visOrangeSID));
          SC_Goalie_Head_Node->addTransition(kicktrans =new
SC_GoalieVisualTarget(SC_Goalie_Kick_Node, visOrangeSID));
          //If you have been shuffling following the ball
for a while, just follow it with your head.
          SC_Goalie_Head_Node->addTransition(new
TimeOutTrans(SC_Pan_Head_Node,2000));




          //After kicking the ball, go back to a pounce
position, but this pounce is a different node than the
first pounce used
          SC_Goalie_Kick_Node->addTransition(new
TimeOutTrans(SC_Goalie_Pounce_Node,3500));
          //SC_Goalie_Kick_Node->addTransition(tmptrans=new
SC_GoalieShuffTrans(SC_Goalie_Shuffle_Node,visOrangeSID));







          //preload the sounds so we don't pause on
tranisitions
          sndman->LoadFile("cutey.wav");
          sndman->LoadFile("barkmed.wav");
          sndman->LoadFile("whimper.wav");
          sndman->LoadFile("fart.wav");
     }



     void SC_Goalie_Lil_Shuff::teardown() {
          //release the sounds
          sndman->ReleaseFile("cutey.wav");
          sndman->ReleaseFile("barkmed.wav");
```

```
          sndman->ReleaseFile("whimper.wav");
          sndman->ReleaseFile("fart.wav");
          StateNode::teardown();
     }

private:
     SC_Goalie_Lil_Shuff(const SC_Goalie_Lil_Shuff&);
          //!< don't call;just satisfies the compiler
     SC_Goalie_Lil_Shuff operator=(const
SC_Goalie_Lil_Shuff&);   //!< don't call;just satisfies the
compiler

};

#endif

// Author    : Ebony Smith
// Date      : 04/2006
// Description:  Main goalie StateNode that implements goal
state machine


//-*-c++-*-
#ifndef INCLUDED_SC_GoalieShuffTrans_h_
#define INCLUDED_SC_GoalieShuffTrans_h_

#include "Events/EventRouter.h"
#include "Events/VisionObjectEvent.h"
#include "Shared/debuget.h"
#include "Shared/WorldState.h"
#include "Shared/ERS210Info.h"
#include "Shared/ERS220Info.h"
#include "Shared/ERS7Info.h"

//! causes a transition when a visual object is "close"
class SC_GoalieShuffTrans : public Transition {
public:
     //!constructor
     SC_GoalieShuffTrans(StateNode* destination, unsigned
int source_id, float threshold=300)
     : Transition("SC_GoalieShuffTrans",destination),
sid(source_id), distanceThreshold(threshold) {}

     //!constructor
     SC_GoalieShuffTrans(const std::string& name,
StateNode* destination, unsigned int source_id, float
threshold=300)
```

```
    : Transition("SC_GoalieShuffTrans",name,destination),
sid(source_id), distanceThreshold(threshold) {}

    //!starts listening for the object specified by the
source id in the constructor
    virtual void DoStart() { Transition::DoStart();
erouter->addListener(this,EventBase::visObjEGID,sid); }

    //!called by StateNode when it becomes inactive - undo
whatever you did in Enable()
    virtual void DoStop() { erouter->removeListener(this);
Transition::DoStop(); }

    //!if the object is "close", calls fire()
    virtual void processEvent(const EventBase& e) {
        cout << ">>>in SC_GoalieShuffTrans:  Event SID is
"<< e.getSourceID() << endl;

        const VisionObjectEvent* ve=dynamic_cast<const
VisionObjectEvent*>(&e);
        ASSERTRET(ve!=NULL,"Casting error");


        float x=ve->getCenterX();
        float y=ve->getCenterY();

        cout << ">>>in SC_GoalieShuffTrans: x is " << x
<< " and y is " << y << endl;

        unsigned int IRDistOffset=-1U;
        //The ERS-7 adds more IR distance sensors, so we
have to
        //break it down by model so we can specify which
one
        /*if(state->robotDesign & WorldState::ERS210Mask)
            IRDistOffset=ERS210Info::IRDistOffset;
        else if(state->robotDesign &
WorldState::ERS220Mask)
            IRDistOffset=ERS220Info::IRDistOffset;
        else if(state->robotDesign &
WorldState::ERS7Mask)*/

        IRDistOffset=ERS7Info::NearIRDistOffset;

        cout << ">>>***in SC_GoalieShuffTrans:  IR sensor
reading is "<< state->sensors[IRDistOffset] << endl;
```

```
          //if(x*x+y*y<0.02f && IRDistOffset!=-1U && state-
>sensors[IRDistOffset]<distanceThreshold) {   NOTE:  take
out the x*x+y*y equation.  Trying to look for a square
          if(state-
>sensors[IRDistOffset]<distanceThreshold) {
               cout << ">>>FIRING SC_GoalieShuffTrans
transition with distance threshold " << distanceThreshold
<< " and IR sensor " <<  state->sensors[IRDistOffset] <<
endl;
               fire();
          }
          }

protected:
     //!Source ID of object to track
     unsigned int sid;

     //!Distance at which to trigger transition, in
millimeters
     float distanceThreshold;
     };


// Author    : Ebony Smith
// Date      : 06/2006
// Description:  Used to transition into a walk that
follows the ball horizontally when the ball is sighted at a
specified distance.

#endif

//-*-c++-*-
#ifndef INCLUDED_SC_GoalieVisualTarget_h_
#define INCLUDED_SC_GoalieVisualTarget_h_

#include "Events/EventRouter.h"
#include "Events/VisionObjectEvent.h"
#include "Shared/debuget.h"
#include "Shared/WorldState.h"
#include "Shared/ERS210Info.h"
#include "Shared/ERS220Info.h"
#include "Shared/ERS7Info.h"

//! causes a transition when a visual object is "close"
class SC_GoalieVisualTarget : public Transition {
public:
     //!constructor
```

```
     SC_GoalieVisualTarget(StateNode* destination, unsigned
int source_id, float threshold=200)
     : Transition("SC_GoalieVisualTarget",destination),
sid(source_id), distanceThreshold(threshold) {}

     //!constructor
     SC_GoalieVisualTarget(const std::string& name,
StateNode* destination, unsigned int source_id, float
threshold=200)
     :
Transition("SC_GoalieVisualTarget",name,destination),
sid(source_id), distanceThreshold(threshold) {}

     //!starts listening for the object specified by the
source id in the constructor
     virtual void DoStart() { Transition::DoStart();
erouter->addListener(this,EventBase::visObjEGID,sid); }

     //!called by StateNode when it becomes inactive - undo
whatever you did in Enable()
     virtual void DoStop() { erouter->removeListener(this);
Transition::DoStop(); }

     //!if the object is "close", calls fire()
     virtual void processEvent(const EventBase& e) {
          cout << ">>>in SC_GoalieVisualTarget:  Event SID
is "<< e.getSourceID() << endl;

          const VisionObjectEvent* ve=dynamic_cast<const
VisionObjectEvent*>(&e);
          ASSERTRET(ve!=NULL,"Casting error");


          float x=ve->getCenterX();
          float y=ve->getCenterY();

          cout << ">>>in SC_GoalieVisualTarget: x is " << x
<< " and y is " << y << endl;

          unsigned int IRDistOffset=-1U;
          //The ERS-7 adds more IR distance sensors, so we
have to
          //break it down by model so we can specify which
one
          /*if(state->robotDesign & WorldState::ERS210Mask)
               IRDistOffset=ERS210Info::IRDistOffset;
          else if(state->robotDesign &
WorldState::ERS220Mask)
```

```
              IRDistOffset=ERS220Info::IRDistOffset;
          else if(state->robotDesign &
WorldState::ERS7Mask)*/

          IRDistOffset=ERS7Info::NearIRDistOffset;

          cout << ">>>***in SC_GoalieVisualTarget:  IR
sensor reading is "<< state->sensors[IRDistOffset] << endl;
          //if(x*x+y*y<0.02f && IRDistOffset!=-1U && state-
>sensors[IRDistOffset]<distanceThreshold) {   NOTE:  take
out the x*x+y*y equation.  Trying to look for a square
          if(state-
>sensors[IRDistOffset]<distanceThreshold) {
               cout << ">>>FIRING SC_GoalieVisualTarget
transition with distance threshold " << distanceThreshold
<< " and IR sensor " <<  state->sensors[IRDistOffset] <<
endl;
               fire();
          }
          }

protected:
     //!Source ID of object to track
     unsigned int sid;

     //!Distance at which to trigger transition, in
millimeters
     float distanceThreshold;
     };


// Author    : Ebony Smith
// Date      :  06/2006
// Description:  Used to transition to into a block/kick
when the ball is close enough to the goal.

#endif

//-*-c++-*-
#ifndef INCLUDED_TimeOutTrans_h_
#define INCLUDED_TimeOutTrans_h_

#include "Behaviors/Transition.h"
#include "Events/EventRouter.h"

//! causes a transition after a specified amount of time
has passed
```

```
/*! If any event parameters are specified, this transition
will listen
 *  for matching events, and if any are received, it will
reset the
 *  timer */
class TimeOutTrans : public Transition {
public:
  //! constructor, specify delay in milliseconds
  TimeOutTrans(StateNode* destination, unsigned int delay)
          :
Transition("TimeOutTrans","TimeOutTrans",destination),
d(delay),
               eventargcount(0),
egid(EventBase::unknownEGID), esid(0),
etid(EventBase::statusETID) {}

  //! constructor, specify delay in milliseconds, if any
events matching given parameters are received, the timer
will be reset
  TimeOutTrans(StateNode* destination, unsigned int delay,
EventBase::EventGeneratorID_t gid)
          :
Transition("TimeOutTrans","TimeOutTrans",destination),
d(delay),
               eventargcount(1), egid(gid), esid(0),
etid(EventBase::statusETID) {}

  //! constructor, specify delay in milliseconds, if any
events matching given parameters are received, the timer
will be reset
  TimeOutTrans(StateNode* destination, unsigned int delay,
EventBase::EventGeneratorID_t gid, unsigned int sid)
          :
Transition("TimeOutTrans","TimeOutTrans",destination),
d(delay),
               eventargcount(2), egid(gid), esid(sid),
etid(EventBase::statusETID) {}

  //! constructor, specify delay in milliseconds, if any
events matching given parameters are received, the timer
will be reset
  TimeOutTrans(StateNode* destination, unsigned int delay,
EventBase::EventGeneratorID_t gid, unsigned int sid,
EventBase::EventTypeID_t tid)
          :
Transition("TimeOutTrans","TimeOutTrans",destination),
d(delay),
```

```
                eventargcount(3), egid(gid), esid(sid),
etid(tid) {}

  //!starts timer
  virtual void DoStart() {
    Transition::DoStart();
    switch (eventargcount) {
    case 1: erouter->addListener(this,egid); break;
    case 2: erouter->addListener(this,egid,esid); break;
    case 3: erouter->addListener(this,egid,esid,etid);
break;
    };
    resetTimer();
  }

  //!stops timer
  virtual void DoStop() {
          erouter->removeListener(this);
          Transition::DoStop();
     }

  //!resets timer
  void resetTimer() {
    // std::cout << "Reset @ " << get_time() << " stop @ "
<< get_time()+d << ' ' << this << std::endl;
    erouter->addTimer(this,0,d,false);
  }

  //!if we receive the timer event, fire()
  virtual void processEvent(const EventBase& e) {
    // std::cout << "Timeout @ " << get_time() << " from "
<< event.getName() << ' ' << this << std::endl;
          if(e.getGeneratorID()==EventBase::timerEGID)
               fire();
          else
               resetTimer();
  }

protected:
  //! constructor, specify delay in milliseconds - use
assignment in your subclass's constructor if you want set
#egid,#esid,#etid (don't forget #eventargcount!)
  TimeOutTrans(const std::string& classname, const
std::string& instancename, StateNode* destination, unsigned
int delay)
          : Transition(classname,instancename,destination),
d(delay),
```

```
              eventargcount(0),
egid(EventBase::unknownEGID), esid(0),
etid(EventBase::statusETID) {}

  //!amount to delay (in milliseconds) before transition
  unsigned int d;

     //!level of specificity of events to listen for
     unsigned int eventargcount;

  EventBase::EventGeneratorID_t egid; //!< the event
generator to listen for
  unsigned int esid; //!< the source to listen for
  EventBase::EventTypeID_t etid; //!< the type to listen
for

};

/*! @file
 * @brief Defines TimeOutTrans, which causes a transition
after a specified amount of time has passed
 * @author Ebony Smith *
 * @Date: 01/2006
 */

#endif

#ifndef INCLUDED_SC_AttackerOrangeBlue_h_
#define INCLUDED_SC_AttackerOrangeBlue_h_

#include "Behaviors/Transition.h"
//#include "Behaviors/Nodes/WalkToTargetNode.h"
#include "Behaviors/Nodes/WalkNode.h"
#include "Behaviors/Demos/ExploreMachine.h"
#include "Behaviors/Transitions/TimeOutTrans.h"
#include "Behaviors/Transitions/VisualTargetTrans.h"
#include "Behaviors/Nodes/OutputNode.h"
#include "Behaviors/Nodes/MotionSequenceNode.h"
#include "Behaviors/Nodes/GroupNode.h"
#include "Sound/SoundManager.h"
#include "Shared/ProjectInterface.h"
//#include "Behaviors/Transitions/VisualTargetCloseTrans.h"
//#include "Behaviors/Nodes/KickNode.h"
#include "Behaviors/Nodes/HeadPointerNode.h"
#include "Behaviors/Transitions/CompletionTrans.h"
#include "Behaviors/StateNode.h"
#include "VisionHeader.h"
#include "GoalieWalkNode.h"
```

```
#include "SC_FindOrangeBlueEvent.h"
#include "Behaviors/Transitions/EventTrans.h"

#include "SC_WalkToTargetFastNode.h"
#include "SC_VisualTargetCloseTrans.h"
#include "SC_Attacker_WalkAround.h"


class SC_AttackerOrangeBlue : public StateNode {
protected:
     StateNode *SC_Start_Node;

public:
     SC_AttackerOrangeBlue() :
StateNode("SC_AttackerOrangeBlue"), SC_Start_Node(NULL) {}

     void SC_AttackerOrangeBlue::DoStart() {
          StateNode::DoStart();
          vrOrangeBall->DoStart(); // *** IMPORTANT: Must
add this to start the Visual Routine to look for the ball!
          SC_Start_Node->DoStart();
     }

     virtual void setup() {
          StateNode::setup();

          // Use Visual Routines to find orange ball
          vrOrangeBall = new SC_FindOrangeBlueEvent();


          GroupNode * SC_Start_Walk_And_Pan = new
GroupNode(getName()+"::SC_Start_Walk_And_Pan");
          addNode(SC_Start_Walk_And_Pan);
          {
               WalkNode * SC_Walk_Forward_Node = new
WalkNode(SC_Start_Walk_And_Pan->getName()+"::TurnInPlace",
150 , 0 , 0);
               SC_Walk_Forward_Node->setVelocity(150,0,0);
               SC_Start_Walk_And_Pan-
>addNode(SC_Walk_Forward_Node);


               LargeMotionSequenceNode * SC_Pan_Head_Node =
new LargeMotionSequenceNode(SC_Start_Walk_And_Pan-
>getName()+"::PanHead","/ms/data/motion/begnattk.mot",true)
;
               SC_Start_Walk_And_Pan-
>addNode(SC_Pan_Head_Node);
```

```
          }

          GroupNode * SC_Attacker2_Kick_Node = new
GroupNode(getName()+"::SC_Attacker2_Kick_Node");
          addNode(SC_Attacker2_Kick_Node);
          {

               LargeMotionSequenceNode * SC_Kick_Node = new
LargeMotionSequenceNode(SC_Attacker2_Kick_Node-
>getName()+"::KickBall","SC_Shkic.mot");
               SC_Attacker2_Kick_Node-
>addNode(SC_Kick_Node);

          }

//        SC_WalkToTargetFastNode * SC_Chase_Node = new
SC_WalkToTargetFastNode(visOrangeSID);
          SC_WalkToTargetFastNode * SC_Chase_Node = new
SC_WalkToTargetFastNode(orangeBallYesSID);
          SC_Chase_Node->setName(getName()+"::Chase");
          addNode(SC_Chase_Node);



          GroupNode * SC_Turn_Straight_Head_Node = new
GroupNode(getName()+"::SC_Turn_Straight_Head_Node");
          addNode(SC_Turn_Straight_Head_Node);
          {
               WalkNode * SC_Turn_360 = new
WalkNode(SC_Turn_Straight_Head_Node-
>getName()+"::SC_Turn_360", 0 , 0 , .85);
               SC_Turn_360->setVelocity(0,0,.85);
               SC_Turn_Straight_Head_Node-
>addNode(SC_Turn_360);


               MediumMotionSequenceNode * panhead = new
MediumMotionSequenceNode(getName()+"::PanHead","/ms/data/mo
tion/turnpan.mot",true);
                  SC_Turn_Straight_Head_Node-
>addNode(panhead);
          }


          GroupNode * SC2_Turn_Straight_Head_Node = new
GroupNode(getName()+"::SC2_Turn_Straight_Head_Node");
          addNode(SC2_Turn_Straight_Head_Node);
          {
```

```
               WalkNode * SC_Turn_360 = new
WalkNode(SC2_Turn_Straight_Head_Node-
>getName()+"::SC_Turn_360", 0 , 0 , .85);
               SC_Turn_360->setVelocity(0,0,.85);
               SC2_Turn_Straight_Head_Node-
>addNode(SC_Turn_360);

               SmallMotionSequenceNode *
SC_Straight_Head_Node = new
SmallMotionSequenceNode(SC2_Turn_Straight_Head_Node-
>getName()+"::PanHead","/ms/data/motion/str8head.pos",true)
;
               SC2_Turn_Straight_Head_Node-
>addNode(SC_Straight_Head_Node);

          }


          GroupNode * SC_Walk_And_Pan = new
GroupNode(getName()+"::SC_Walk_And_Pan");
          addNode(SC_Walk_And_Pan);
          {
               WalkNode * SC_Forward_Node = new
WalkNode(SC_Walk_And_Pan->getName()+"::SC_Forward_Node",
150 , 0 , 0);
               SC_Forward_Node->setVelocity(150,0,0);
               SC_Walk_And_Pan->addNode(SC_Forward_Node);


               LargeMotionSequenceNode * SC_Pan_Node = new
LargeMotionSequenceNode(SC_Walk_And_Pan-
>getName()+"::SC_Pan_Node","/ms/data/motion/newpan.mot",tru
e);
               SC_Walk_And_Pan->addNode(SC_Pan_Node);
          }



          WalkNode * SC_360 = new
WalkNode(getName()+"::SC_360", 0 , 0 , .85);
          SC_360->setVelocity(0,0,.85);
          addNode(SC_360);


          WalkNode * SC_180 = new
WalkNode(getName()+"::SC_360", 0 , 0 , .75);
          SC_180->setVelocity(0,0,.75);
          addNode(SC_180);
```

```
          WalkNode * SC2_360 = new
WalkNode(getName()+"::SC2_360", 0 , 0 , .85);
          SC2_360->setVelocity(0,0,.85);
          addNode(SC2_360);


          SmallMotionSequenceNode * SC_Straight_Node = new
SmallMotionSequenceNode(getName()+"::PanHead","/ms/data/mot
ion/atkkpana.pos",true);
          addNode(SC_Straight_Node);

          HeadPointerNode * frontNode = new
HeadPointerNode("UpNode");
          addNode(frontNode);
          frontNode-> getMC()->lookAtPoint(200,0,100,30);


          HeadPointerNode * downNode = new
HeadPointerNode("downNode");
          addNode(downNode);
          downNode-> getMC()->lookAtPoint(200,0,30,30);


          GroupNode * SC_Pan_Pounce_2_Node = new
GroupNode(getName()+"::SC_Pan_Pounce_2_Node");
          addNode(SC_Pan_Pounce_2_Node);
          {
               LargeMotionSequenceNode * SC_Pan_Only_Node =
new LargeMotionSequenceNode(SC_Pan_Pounce_2_Node-
>getName()+"::PanHead","/ms/data/motion/newpan.mot",true);
               SC_Pan_Pounce_2_Node-
>addNode(SC_Pan_Only_Node);

               StateNode *SC_Pounce_Node= new
MediumMotionSequenceNode(SC_Pan_Pounce_2_Node-
>getName()+"::stand","/ms/data/motion/pounce.pos",false);
               SC_Pan_Pounce_2_Node-
>addNode(SC_Pounce_Node);

          }
          StateNode *SC_Test_Node= new
MediumMotionSequenceNode(getName()+"::stand","/ms/data/moti
on/pounce.pos",false);
          addNode(SC_Test_Node);
```

```
          Transition * tmptrans=NULL;
          Transition * kicktrans=NULL;
          //Transition * ctrans=NULL;


          //starts out exploring
          SC_Start_Node=SC_Start_Walk_And_Pan;
          //SC_Start_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Chase_Node,visOrangeSID));
          // CHANGED this to a different kind of event
//        SC_Start_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Chase_Node, orangeBallYesSID));
          SC_Start_Walk_And_Pan->addTransition(new
EventTrans(SC_Chase_Node, EventBase::visObjEGID,
orangeBallYesSID, EventBase::activateETID)); // Need this
event transition for orange ball

//        SC_Start_Walk_And_Pan->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,visOrangeS
ID, 160));


          SC_Start_Walk_And_Pan->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
          SC_Start_Walk_And_Pan->addTransition(new
TimeOutTrans(frontNode,3500));
          //SC_Start_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));
          //SC_Start_Walk_And_Pan->addTransition(new
SC_VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID,
160));


          //SC_Chase_Node Transistions
//        SC_Chase_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,visOrangeS
ID, 160));

          //Test for turn away for defending goal
          SC_Chase_Node->addTransition(new
EventTrans(SC_180, EventBase::visObjEGID, orangeBlueYesSID,
EventBase::activateETID)); // Need this event transition
for Blue goal and orange ball
          SC_Chase_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
```

```
          SC_Chase_Node-
>addTransition(tmptrans=SC_Chase_Node-
>newDefaultLostTrans(SC_Pan_Pounce_2_Node));
          //SC_Chase_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID,
160));


          //SC_Attacker2_Kick_Node Transistions
          SC_Attacker2_Kick_Node->addTransition(new
TimeOutTrans(SC_Pan_Pounce_2_Node,1000));

          //SC_Pan_Pounce_2_Node Transistions


          SC_Pan_Pounce_2_Node->addTransition(new
TimeOutTrans(frontNode,2500));
          SC_Pan_Pounce_2_Node->addTransition(new
VisualTargetTrans(SC_Chase_Node,orangeBallYesSID));
          SC_Pan_Pounce_2_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
          //SC_Pan_Pounce_2_Node->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));
          //SC_Pan_Pounce_2_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID,
160));

          frontNode->addTransition(new
TimeOutTrans(SC_360,500));

          //SC_Turn_360 Transitions

          SC_360->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
          SC_360->addTransition(new
VisualTargetTrans(SC_Chase_Node,orangeBallYesSID));
          SC_360->addTransition(new
TimeOutTrans(downNode,6000));


//        SC_180->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
//        SC_180->addTransition(new
VisualTargetTrans(SC_Chase_Node,orangeBallYesSID));
```

```
          SC_180->addTransition(new
TimeOutTrans(SC_Pan_Pounce_2_Node,2500));
          //SC_360->addTransition(new
SC_VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID,
160));
          //SC_360->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));


          downNode->addTransition(new
TimeOutTrans(SC_Turn_Straight_Head_Node,500));


          SC_Turn_Straight_Head_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
          SC_Turn_Straight_Head_Node->addTransition(new
VisualTargetTrans(SC_Chase_Node,orangeBallYesSID));
          SC_Turn_Straight_Head_Node->addTransition(new
TimeOutTrans(SC_Walk_And_Pan,6000));
          //SC_Turn_Straight_Head_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID,
160));
          //SC_Turn_Straight_Head_Node->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));




          SC_Walk_And_Pan->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
          SC_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Chase_Node,orangeBallYesSID));
          SC_Walk_And_Pan->addTransition(new
TimeOutTrans(frontNode,3000));
          //SC_Walk_And_Pan->addTransition(new
SC_VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID,
160));
          //SC_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));




          //preload the sounds so we don't pause on
tranisitions
          sndman->LoadFile("cutey.wav");
```

```
            sndman->LoadFile("barkmed.wav");
            sndman->LoadFile("whimper.wav");
            sndman->LoadFile("fart.wav");
        }



        void SC_AttackerOrangeBlue::teardown() {
            //release the sounds
            sndman->ReleaseFile("cutey.wav");
            sndman->ReleaseFile("barkmed.wav");
            sndman->ReleaseFile("whimper.wav");
            sndman->ReleaseFile("fart.wav");
            StateNode::teardown();
        }

    private:
        SC_FindOrangeBlueEvent *vrOrangeBall;

        SC_AttackerOrangeBlue(const SC_AttackerOrangeBlue&);
                    //!< don't call;just satisfies the compiler
        SC_AttackerOrangeBlue operator=(const
    SC_AttackerOrangeBlue&);      //!< don't call;just
    satisfies the compiler

    };

    #endif

    #ifndef INCLUDED_SC_AttackerOrangeYellow_h_
    #define INCLUDED_SC_AttackerOrangeYellow_h_

    #include "Behaviors/Transition.h"
    //#include "Behaviors/Nodes/WalkToTargetNode.h"
    #include "Behaviors/Nodes/WalkNode.h"
    #include "Behaviors/Demos/ExploreMachine.h"
    #include "Behaviors/Transitions/TimeOutTrans.h"
    #include "Behaviors/Transitions/VisualTargetTrans.h"
    #include "Behaviors/Nodes/OutputNode.h"
    #include "Behaviors/Nodes/MotionSequenceNode.h"
    #include "Behaviors/Nodes/GroupNode.h"
    #include "Sound/SoundManager.h"
    #include "Shared/ProjectInterface.h"
    //#include "Behaviors/Transitions/VisualTargetCloseTrans.h"
    //#include "Behaviors/Nodes/KickNode.h"
    #include "Behaviors/Nodes/HeadPointerNode.h"
    #include "Behaviors/Transitions/CompletionTrans.h"
    #include "Behaviors/StateNode.h"
```

```
#include "VisionHeader.h"
#include "GoalieWalkNode.h"
#include "SC_FindOrangeYellowEvent.h"
#include "Behaviors/Transitions/EventTrans.h"

#include "SC_WalkToTargetFastNode.h"
#include "SC_VisualTargetCloseTrans.h"
#include "SC_Attacker_WalkAround.h"


class SC_AttackerOrangeYellow : public StateNode {
protected:
     StateNode *SC_Start_Node;

public:
     SC_AttackerOrangeYellow() :
StateNode("SC_AttackerOrangeYellow"), SC_Start_Node(NULL)
{}

     void SC_AttackerOrangeYellow::DoStart() {
          StateNode::DoStart();
          vrOrangeBall->DoStart(); // *** IMPORTANT: Must
add this to start the Visual Routine to look for the ball!
          SC_Start_Node->DoStart();
     }

     virtual void setup() {
          StateNode::setup();

          // Use Visual Routines to find orange ball
          vrOrangeBall = new SC_FindOrangeYellowEvent();


          GroupNode * SC_Start_Walk_And_Pan = new
GroupNode(getName()+"::SC_Start_Walk_And_Pan");
          addNode(SC_Start_Walk_And_Pan);
          {
               WalkNode * SC_Walk_Forward_Node = new
WalkNode(SC_Start_Walk_And_Pan->getName()+"::TurnInPlace",
150 , 0 , 0);
               SC_Walk_Forward_Node->setVelocity(150,0,0);
               SC_Start_Walk_And_Pan-
>addNode(SC_Walk_Forward_Node);


               LargeMotionSequenceNode * SC_Pan_Head_Node =
new LargeMotionSequenceNode(SC_Start_Walk_And_Pan-
```

```
>getName()+"::PanHead","/ms/data/motion/begnattk.mot",true)
;
                SC_Start_Walk_And_Pan-
>addNode(SC_Pan_Head_Node);
          }

          GroupNode * SC_Attacker2_Kick_Node = new
GroupNode(getName()+"::SC_Attacker2_Kick_Node");
          addNode(SC_Attacker2_Kick_Node);
          {

                LargeMotionSequenceNode * SC_Kick_Node = new
LargeMotionSequenceNode(SC_Attacker2_Kick_Node-
>getName()+"::KickBall","SC_Shkic.mot");
                SC_Attacker2_Kick_Node-
>addNode(SC_Kick_Node);

          }

//        SC_WalkToTargetFastNode * SC_Chase_Node = new
SC_WalkToTargetFastNode(visOrangeSID);
          SC_WalkToTargetFastNode * SC_Chase_Node = new
SC_WalkToTargetFastNode(orangeBallYesSID);
          SC_Chase_Node->setName(getName()+"::Chase");
          addNode(SC_Chase_Node);



          GroupNode * SC_Turn_Straight_Head_Node = new
GroupNode(getName()+"::SC_Turn_Straight_Head_Node");
          addNode(SC_Turn_Straight_Head_Node);
          {
                WalkNode * SC_Turn_360 = new
WalkNode(SC_Turn_Straight_Head_Node-
>getName()+"::SC_Turn_360", 0 , 0 , .85);
                SC_Turn_360->setVelocity(0,0,.85);
                SC_Turn_Straight_Head_Node-
>addNode(SC_Turn_360);


                MediumMotionSequenceNode * panhead = new
MediumMotionSequenceNode(getName()+"::PanHead","/ms/data/mo
tion/turnpan.mot",true);
                  SC_Turn_Straight_Head_Node-
>addNode(panhead);
          }
```

```
          GroupNode * SC2_Turn_Straight_Head_Node = new
GroupNode(getName()+"::SC2_Turn_Straight_Head_Node");
          addNode(SC2_Turn_Straight_Head_Node);
          {
               WalkNode * SC_Turn_360 = new
WalkNode(SC2_Turn_Straight_Head_Node-
>getName()+"::SC_Turn_360", 0 , 0 , .85);
               SC_Turn_360->setVelocity(0,0,.85);
               SC2_Turn_Straight_Head_Node-
>addNode(SC_Turn_360);

               SmallMotionSequenceNode *
SC_Straight_Head_Node = new
SmallMotionSequenceNode(SC2_Turn_Straight_Head_Node-
>getName()+"::PanHead","/ms/data/motion/str8head.pos",true)
;
               SC2_Turn_Straight_Head_Node-
>addNode(SC_Straight_Head_Node);

          }


          GroupNode * SC_Walk_And_Pan = new
GroupNode(getName()+"::SC_Walk_And_Pan");
          addNode(SC_Walk_And_Pan);
          {
               WalkNode * SC_Forward_Node = new
WalkNode(SC_Walk_And_Pan->getName()+"::SC_Forward_Node",
150 , 0 , 0);
               SC_Forward_Node->setVelocity(150,0,0);
               SC_Walk_And_Pan->addNode(SC_Forward_Node);


               LargeMotionSequenceNode * SC_Pan_Node = new
LargeMotionSequenceNode(SC_Walk_And_Pan-
>getName()+"::SC_Pan_Node","/ms/data/motion/newpan.mot",tru
e);
               SC_Walk_And_Pan->addNode(SC_Pan_Node);
          }



          WalkNode * SC_360 = new
WalkNode(getName()+"::SC_360", 0 , 0 , .85);
          SC_360->setVelocity(0,0,.85);
          addNode(SC_360);
```

```
          WalkNode * SC_180 = new
WalkNode(getName()+"::SC_360", 0 , 0 , .75);
          SC_180->setVelocity(0,0,.75);
          addNode(SC_180);

          WalkNode * SC2_360 = new
WalkNode(getName()+"::SC2_360", 0 , 0 , .85);
          SC2_360->setVelocity(0,0,.85);
          addNode(SC2_360);


          SmallMotionSequenceNode * SC_Straight_Node = new
SmallMotionSequenceNode(getName()+"::PanHead","/ms/data/mot
ion/atkkpana.pos",true);
          addNode(SC_Straight_Node);

          HeadPointerNode * frontNode = new
HeadPointerNode("UpNode");
          addNode(frontNode);
          frontNode-> getMC()->lookAtPoint(200,0,100,30);


          HeadPointerNode * downNode = new
HeadPointerNode("downNode");
          addNode(downNode);
          downNode-> getMC()->lookAtPoint(200,0,30,30);


          GroupNode * SC_Pan_Pounce_2_Node = new
GroupNode(getName()+"::SC_Pan_Pounce_2_Node");
          addNode(SC_Pan_Pounce_2_Node);
          {
               LargeMotionSequenceNode * SC_Pan_Only_Node =
new LargeMotionSequenceNode(SC_Pan_Pounce_2_Node-
>getName()+"::PanHead","/ms/data/motion/newpan.mot",true);
               SC_Pan_Pounce_2_Node-
>addNode(SC_Pan_Only_Node);

               StateNode *SC_Pounce_Node= new
MediumMotionSequenceNode(SC_Pan_Pounce_2_Node-
>getName()+"::stand","/ms/data/motion/pounce.pos",false);
               SC_Pan_Pounce_2_Node-
>addNode(SC_Pounce_Node);

          }
          StateNode *SC_Test_Node= new
MediumMotionSequenceNode(getName()+"::stand","/ms/data/moti
on/pounce.pos",false);
```

```
          addNode(SC_Test_Node);



          Transition * tmptrans=NULL;
          Transition * kicktrans=NULL;
          //Transition * ctrans=NULL;


          //starts out exploring
          SC_Start_Node=SC_Start_Walk_And_Pan;
          //SC_Start_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Chase_Node,visOrangeSID));
          // CHANGED this to a different kind of event
//        SC_Start_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Chase_Node, orangeBallYesSID));
          SC_Start_Walk_And_Pan->addTransition(new
EventTrans(SC_Chase_Node, EventBase::visObjEGID,
orangeBallYesSID, EventBase::activateETID)); // Need this
event transition for orange ball

//        SC_Start_Walk_And_Pan->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,visOrangeS
ID, 160));


          SC_Start_Walk_And_Pan->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
          SC_Start_Walk_And_Pan->addTransition(new
TimeOutTrans(frontNode,3500));
          //SC_Start_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));
          //SC_Start_Walk_And_Pan->addTransition(new
SC_VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID,
160));


          //SC_Chase_Node Transistions
//        SC_Chase_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,visOrangeS
ID, 160));

          //Test for turn away for defending goal
          SC_Chase_Node->addTransition(new
EventTrans(SC_180, EventBase::visObjEGID,
orangeYellowYesSID, EventBase::activateETID)); // Need this
event transition for yellow goal and orange ball
```

```
          SC_Chase_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));

          SC_Chase_Node-
>addTransition(tmptrans=SC_Chase_Node-
>newDefaultLostTrans(SC_Pan_Pounce_2_Node));
          //SC_Chase_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID,
160));


          //SC_Attacker2_Kick_Node Transistions
          SC_Attacker2_Kick_Node->addTransition(new
TimeOutTrans(SC_Pan_Pounce_2_Node,1000));

          //SC_Pan_Pounce_2_Node Transistions


          SC_Pan_Pounce_2_Node->addTransition(new
TimeOutTrans(frontNode,2500));
          SC_Pan_Pounce_2_Node->addTransition(new
VisualTargetTrans(SC_Chase_Node,orangeBallYesSID));
          SC_Pan_Pounce_2_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
          //SC_Pan_Pounce_2_Node->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));
          //SC_Pan_Pounce_2_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID,
160));

          frontNode->addTransition(new
TimeOutTrans(SC_360,500));

          //SC_Turn_360 Transitions

          SC_360->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
          SC_360->addTransition(new
VisualTargetTrans(SC_Chase_Node,orangeBallYesSID));
          SC_360->addTransition(new
TimeOutTrans(downNode,6000));
```

```
//        SC_180->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
//        SC_180->addTransition(new
VisualTargetTrans(SC_Chase_Node,orangeBallYesSID));
          SC_180->addTransition(new
TimeOutTrans(SC_Pan_Pounce_2_Node,2500));
          //SC_360->addTransition(new
SC_VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID,
160));
          //SC_360->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));


          downNode->addTransition(new
TimeOutTrans(SC_Turn_Straight_Head_Node,500));


          SC_Turn_Straight_Head_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
          SC_Turn_Straight_Head_Node->addTransition(new
VisualTargetTrans(SC_Chase_Node,orangeBallYesSID));
          SC_Turn_Straight_Head_Node->addTransition(new
TimeOutTrans(SC_Walk_And_Pan,6000));
          //SC_Turn_Straight_Head_Node->addTransition(new
SC_VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID,
160));
          //SC_Turn_Straight_Head_Node->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));




          SC_Walk_And_Pan->addTransition(new
SC_VisualTargetCloseTrans(SC_Attacker2_Kick_Node,orangeBall
YesSID, 160));
          SC_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Chase_Node,orangeBallYesSID));
          SC_Walk_And_Pan->addTransition(new
TimeOutTrans(frontNode,3000));
          //SC_Walk_And_Pan->addTransition(new
SC_VisualTargetCloseTrans(SC_Test_Node,yellowGoalSID,
160));
          //SC_Walk_And_Pan->addTransition(new
VisualTargetTrans(SC_Test_Node,yellowGoalSID));
```

```
          //preload the sounds so we don't pause on
tranisitions
          sndman->LoadFile("cutey.wav");
          sndman->LoadFile("barkmed.wav");
          sndman->LoadFile("whimper.wav");
          sndman->LoadFile("fart.wav");
     }



     void SC_AttackerOrangeYellow::teardown() {
          //release the sounds
          sndman->ReleaseFile("cutey.wav");
          sndman->ReleaseFile("barkmed.wav");
          sndman->ReleaseFile("whimper.wav");
          sndman->ReleaseFile("fart.wav");
          StateNode::teardown();
     }

private:
     SC_FindOrangeYellowEvent *vrOrangeBall;

     SC_AttackerOrangeYellow(const
SC_AttackerOrangeYellow&);                    //!< don't
call;just satisfies the compiler
     SC_AttackerOrangeYellow operator=(const
SC_AttackerOrangeYellow&);    //!< don't call;just
satisfies the compiler

};

#endif
```