Disclaimer: A report submitted to Dublin City University, School of Computing for the 4th Year Project, 2009/2010. I hereby certify that the work presented and the material contained herein is my own except where explicitly stated references to other material are made.

DS1

SwitchBall

User Manual

Eoin Costelloe (56586261)

4th Year Project – DCU May 2010

Supervisor : Mr David Sinclair

Introduction:

This game is a basketball based game where two teams compete to score goals on each other. Pass the ball to team members and try to find weaknesses in the opposition's defence. After the time limit has been reached, the results of the match are given.

How to Run the Program:

After downloading and unzipping the game (which is freely available from the website http://fluffymachappy.com/Game/SwitchBall.zip), you are all set up. There is no installation involved once you have the necessary files downloaded. To run the game, double-click SwitchBall.jar which is located within the folder SwitchBall. The game requires Java 2 Standard Edition (J2SE) version 1.4 or later to run.

Main Menu (Figure 1):

Once the program has been started, you are given 4 options to choose from:

- You can dive right into the game and leave everything at default by selecting option 1, Quick Match.
- You can define your own settings for the game by selecting option 2, Player Match.
- You can create your own team to play with in option 3, Team Builder.
- If you are confused about the controls of the game then select option 4, Controls.

Quick Match:

In a Quick Match you can get right into the action. Team 1 is Humans, team 2 is Flying. The time limit is set to 60 seconds.

Player Match (Figure 2):

Here you are given several options which you can choose from to define your game, which include: the game map, team 1, team 2, the difficulty of the AI team members, and also the time limit. Experiment with these settings to find a game that suits your style best.

Team Builder (Figure 3):

This allows you to create your own team or modify an already-existing team. You are given several players with different unique attributes. Try to balance your team to get the most out of it. Your own unique team is just waiting to be created, so be creative.

Controls (Figure 4):

This is a handy reminder of how to control your player and other actions you can do while playing the game.

Playing SwitchBall:

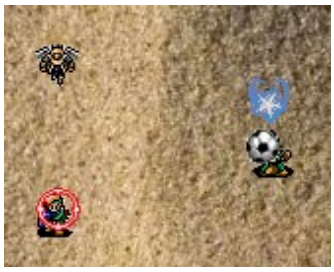
I hope you are ready to put yourself at the test. Create your own custom team and compete with friends in this challenging and addictive game. Be the best and beat the rest. Each player on your team has unique attributes that give them an edge such as speed and strength, so try to take advantage of the opponent's weaknesses using your strengths. Last but not least, try to score as many goals against your opponent within the time-frame to become the winner.

Speed:

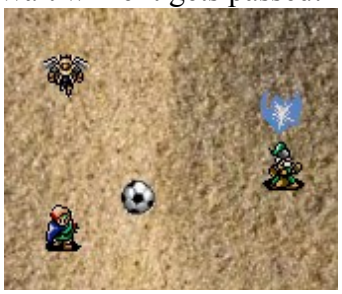
Each player can move around the playing field, but some players are faster than others so watch out and keep an eye on the distances between players.

Passing / Shooting:

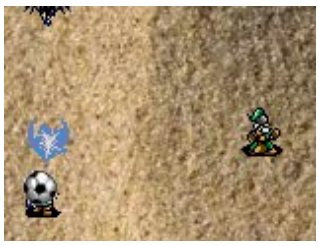
Choose where to pass the ball:



Wait while it gets passed:



Catch the ball:



Be careful when passing the ball or shooting. Each player is given a strength and this determines the strength they can pass the ball with. The longer the ball is sent through the air, the weaker it becomes. If the ball's strength is too weak, the ball is fumbled and the closest opposing player is given control of it.

Tackle:

Each opposing player can tackle you if you have the ball. This forces you to pass the ball to one of your team members. Try to use tackling to stop opposing team members from running around your defence. Also remember that each player has a different tackle range and so are better at defending than others.

Interception:

Try to get between the opposing player and where he is passing it:

Intercept the ball as it passes by your player:

Gain control of the ball:

When passing the ball, opposing team members can reduce the ball's strength as it passes through their interception range. A greater interception range means a greater chance that you will intercept the ball, but a greater interception strength will reduce the ball's strength more quickly. This can be very useful if you want to block off passing between opposing players.

Overview of Controls:

Left, Right, Up and Down keys are player 1 movement.
Enter key is player 1 pass or shoot.
W, A, S and D keys are player 2 movement.
Space key is player 2 pass or shoot.
P key is pause.
Escape key exits the game.

Figure 1:

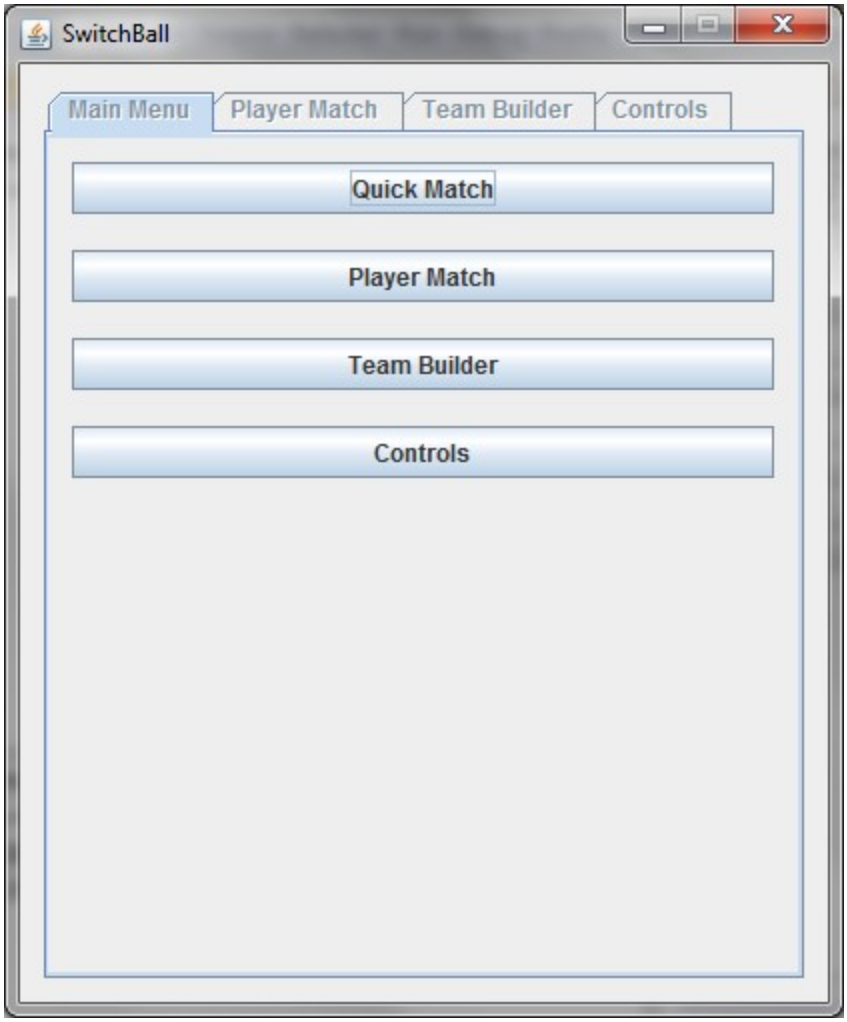

SwitchBall
Main Menu
Player Match
Team Builder
Controls
Quick Match
Player Match
Team Builder
Controls

Figure 2:

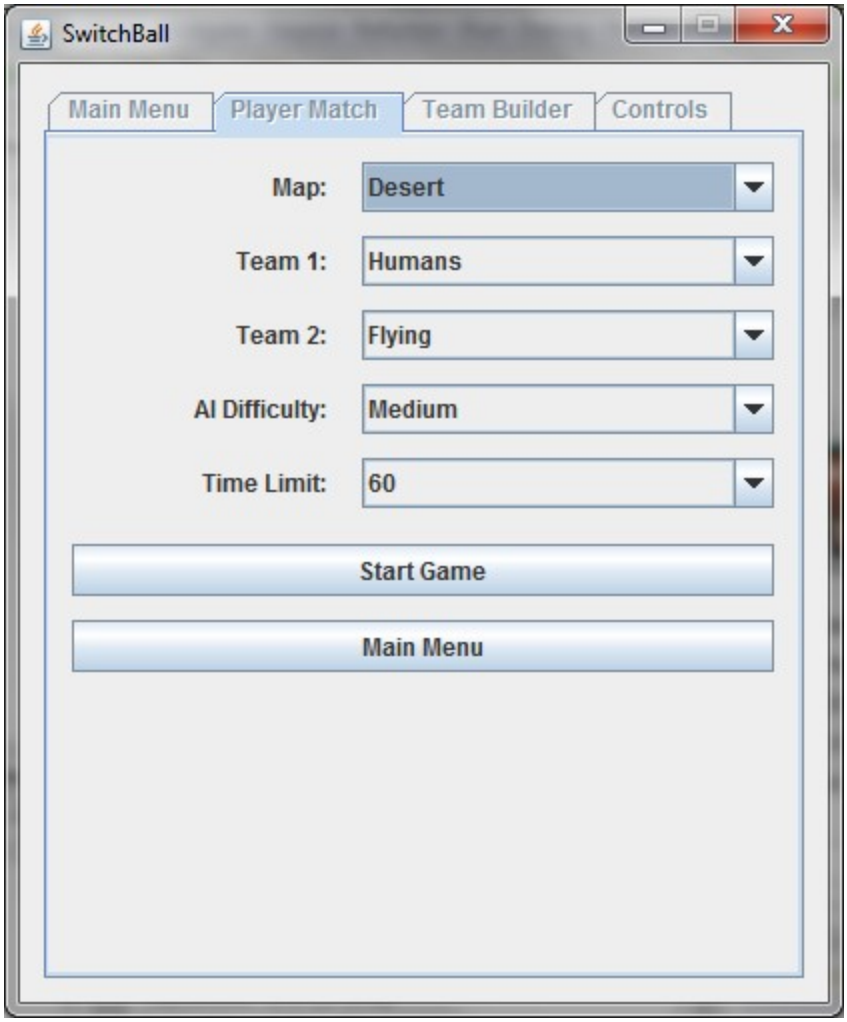

SwitchBall
Main Menu
Player Match
Team Builder
Controls
Map:
Desert
Team 1:
Humans
Team 2:
Flying
AI Difficulty:
Medium
Time Limit:
60
Start Game
Main Menu

Figure 3:

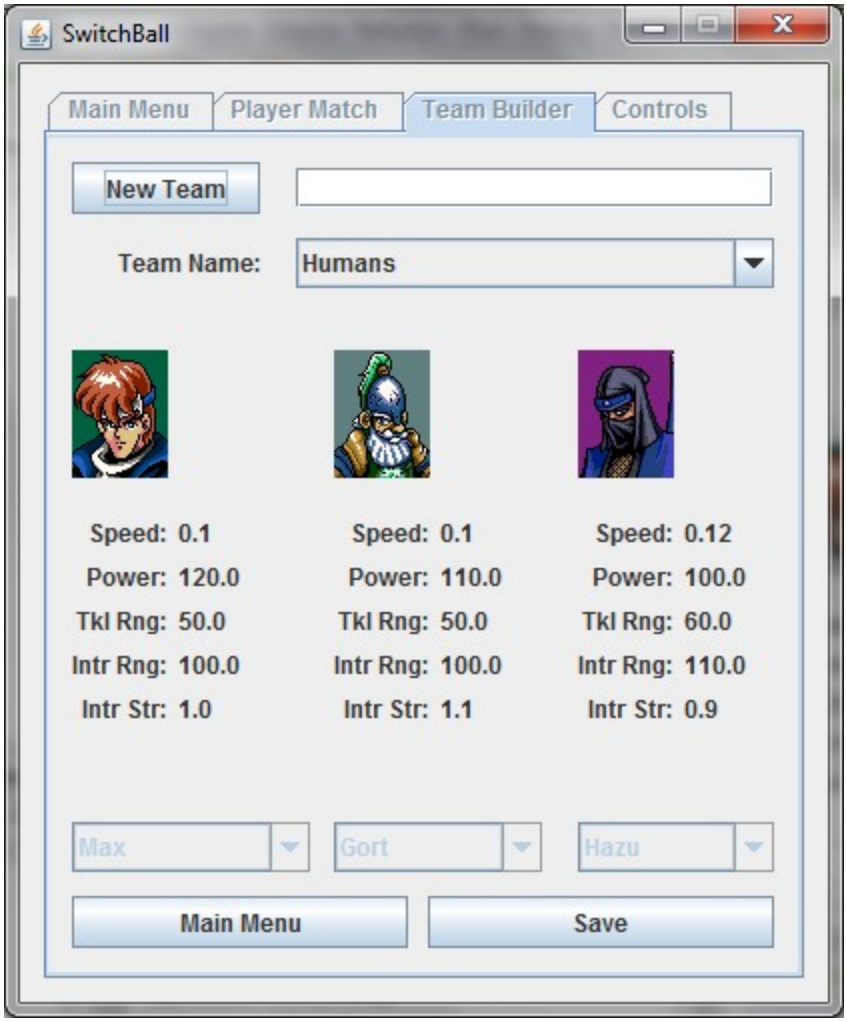

SwitchBall
Main Menu
Player Match
Team Builder
Controls
New Team
Team Name:
Humans
Speed: 0.1
Power: 120.0
Tkl Rng: 50.0
Intr Rng: 100.0
Intr Str: 1.0
Speed: 0.1
Power: 110.0
Tkl Rng: 50.0
Intr Rng: 100.0
Intr Str: 1.1
Speed: 0.12
Power: 100.0
Tkl Rng: 60.0
Intr Rng: 110.0
Intr Str: 0.9
Max
Gort
Hazu
Main Menu
Save

Figure 4:


SwitchBall
Main Menu
Player Match
Team Builder
Controls
Player 1:
Keyboard
Move Left: Left Key
Move Up: Up Key
Move Right: Right Key
Move Down: Down Key
Pass/Shoot: Enter Key
Player 2:
Keyboard
Move Left: A Key
Move Up: W Key
Move Right: D Key
Move Down: S Key
Pass/Shoot: Space Key
Pause Game: P Key
Exit Game: Escape Key
Main Menu

Disclaimer: A report submitted to Dublin City University, School of Computing for the 4th Year Project, 2009/2010. I hereby certify that the work presented and the material contained herein is my own except where explicitly stated references to other material are made.

DS1

SwitchBall

Technical Specification

Eoin Costelloe (56586261)

4th Year Project – DCU May 2010

Supervisor : Mr David Sinclair

Table of Contents

1 Introduction

1.1 Overview

This product is a basketball-like computer game where two teams compete to score goals. Each team will have three players. Each player has specific attributes which include speed, passing power, tackle range, interception range and interception strength. Each match ends when the time limit is reached, which is set by the user. The results of the match are displayed when the match is finished.

This is a two-user game. The player on the team with the ball is controlled by the user where he can move around the field and other team members are controlled by AI. The user can then pass the ball to other players. As the ball is passed, its strength gets weaker. When the ball is passed, it can be intercepted by opposing team members which decreases the strength of the passing ball. The player can also shoot at the goals which has the same principle as passing.

This game can either be played as a stand-alone application or embedded into a website. It will therefore be competing with a large amount of other online and offline games. This game will offer the ease of use of a simple 2-dimensional game with the complexity of a well-built AI. It will also have enough additional functionality to keep the user entertained, such as multiplayer and customizable teams.

1.2 Glossary

AI: Artificial Intelligence

API: Application Programming Interface

DOM (Document Object Model): an internal way of storing and searching through a hierarchy of data

GTGE (Golden T Game Engine): an advanced cross-platform game programming library

GUI: Graphical User Interface

IDE: Integrated Development Environment

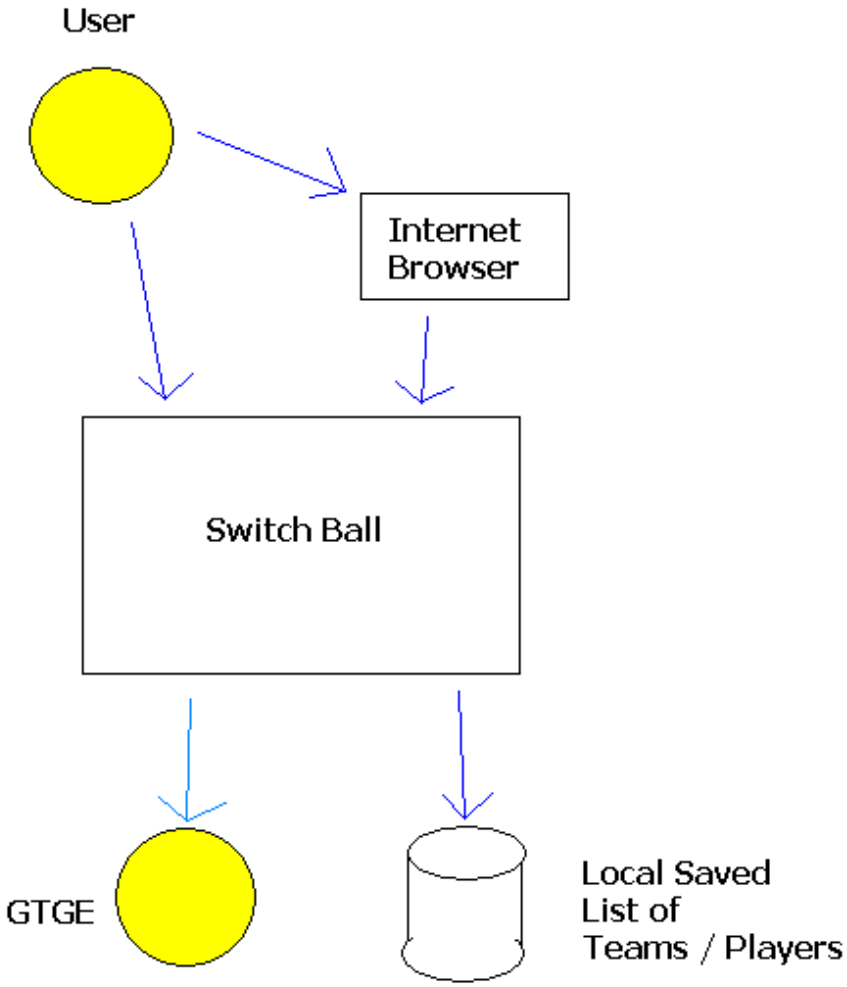
JVM: Java Virtual Machine

RMI (Remote Method Invocation): allows the use of Remote Procedure Calls in a distributed system

XML (Extensible Markup Language): a general purpose specification for creating custom markup languages

2 System Architecture Diagram

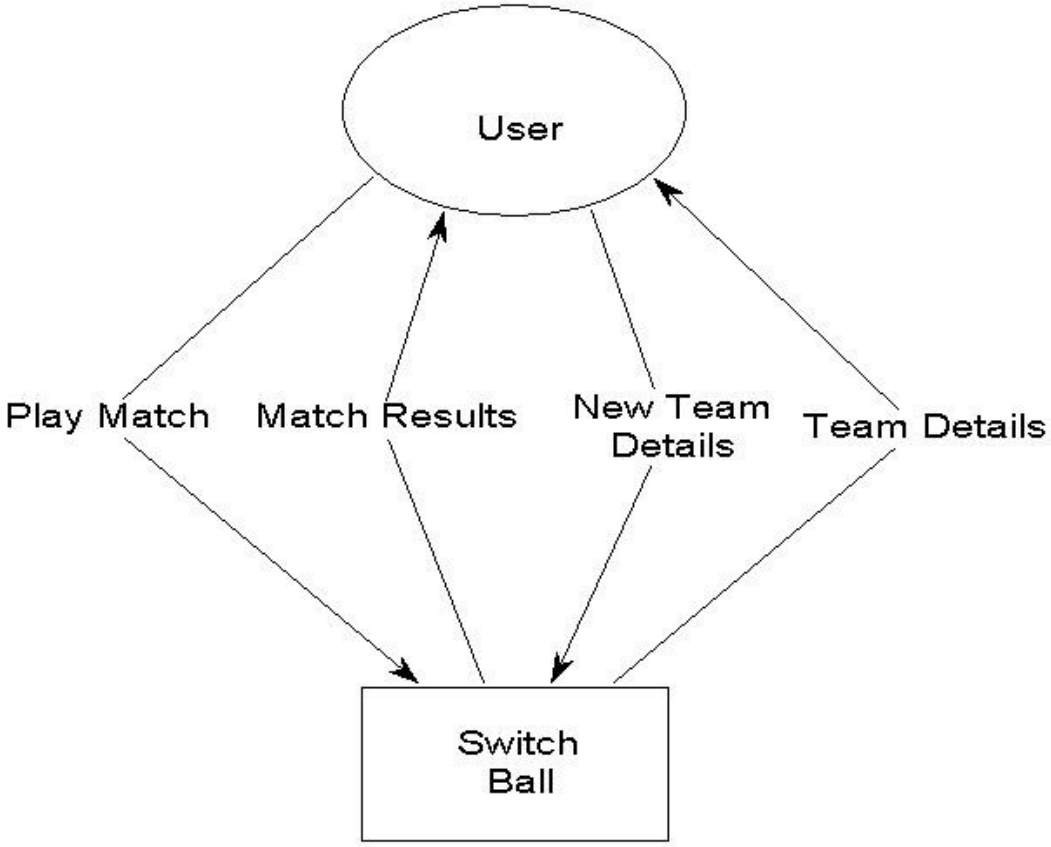
- The user can access the program either by directly downloading and running the program locally or by running it online through an internet browser.
- The program then accesses a locally stored list of teams and players.



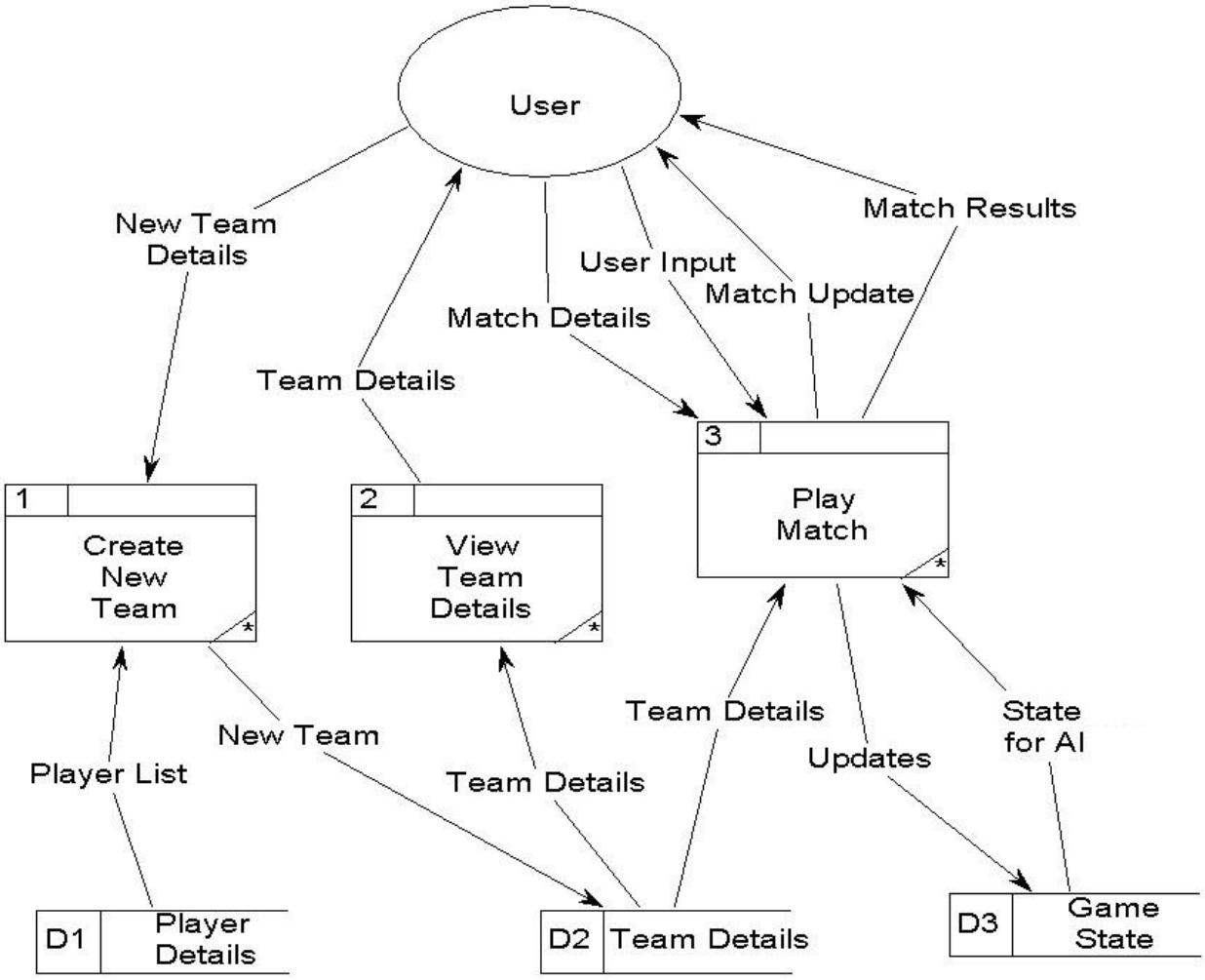
3 High Level Design

Our program interacts with the user while he is in the menu, and with two users during the game. This interaction is identified in the context diagram. All the data and processes in our program are detailed in our level 1 data flow diagram.

3.1 Context Diagram:



3.2 Level 1 Data Flow Diagram:



4 Problems, Challenges and Resolutions

Game Resources

As this is a computer game, we must find user friendly and open source content for this game. It was decided to use Netbeans for the main menu. I also found a good source for freely-available sprites at the Sprite Database.

Player Match User Modifications

Given the time constraints and complexity, we cannot allow the user to have full control over every aspect of the game. The program allows the user to choose the map, team 1, team 2, AI Difficulty and the time limit. We could not allow the user to change the game's resolution, for example, as the code to make it multi-platform and allow for multiple resolutions would not be cost-efficient for the testing and documentation involved.

Marking User Players and Player with Ball

A big problem when designing the look of the game was to show the location of the player that the user is in control of in a nice and easy way. This is done using a marker over their head. We cannot use the same tactic however on showing which player has the ball, as this would cause too many markers over the player's head. We mark the player with the ball by overlaying the ball on top of the player. Although this covers most of the player's image, it is more important to define where the ball is to the user.

Outputting XML to a file

We can use the default Document Builder which is built into Java (javax.xml) to input the XML file and store it internally but to output the DOM, we use external class (com.sun.org.apache.xml). This reduces the amount of components my program has to test and manage. It also increases the amount of third party software being used in my program.

Defining an Artificial Intelligence Model

We have to implement an Artificial Intelligence Model that allows easy additions to different moods which allow different tactics by the AI players. I decided on a sub-symbolic approach as we are already aware of good tactics that are present in basketball and this type of approach is easier to test and implement than its counterparts. I have also dealt with this type of AI before in previous DCU Modules.

Displaying Game State

After user testing, it was decided that not enough information about the game state was given to the user. A display window was incorporated into the side of the game which shows the current state of the game to the user. This shows the player who has the ball, the power of the ball as it is being passed and anything else that the user may need to know.

Faulty Exiting the GTGE Game

It was found after lots of research and testing with the help of the developers of GTGE (http://goldenstudios.or.id/forum/showthread.php?tid=3220) that System.exit(0) was being called as the GTGE game thread was being closed. This would cause the whole process to exit which includes the main menu. This was solved by overriding the protected method notify() and removing the call to System.exit(0).

5 Additional Features:

This program could add in online multiplayer capabilities over the network. Most of the processing (such as AI) would be done on the server and sent to the client. This could also be linked to a game lobby using RMI where the user could easily select another online game to join. The traffic between client and server would be plain-text XML.

The program could also store extra team data alongside the list of players. This team data would include their amount of wins and losses. This could be used to create a leader board system.

We could define a more descriptive artificial intelligence which allows the computer to gain control of the ball. This could allow the user to play against the computer in a single player game. However, under the time constraints, it is not feasible to create a properly-tested and documented deliverable by the deadline.

The user could play against the computer in randomly-designed tournaments based upon the past results. This may also be linked to an online tournament where users can compete in ranked tournaments. The objective of a ranked tournament is to link users of equal skill based on their past statistics.

The program could modify user input to allow customisable controls. This could be from a game controller or other external devices. This should make the game more usable and customizable.

6 Appendices

Application tools and Links:

- http://www.w3.org/xml: this is an in-depth explanation of how XML is incorporated into a network
- http://goldenstudios.or.id/products/GTGE/: an open source high level API for generating 2 dimensional games
- http://netbeans.org/: this a nice IDE which creates easy to use and implement GUIs
- http://java.sun.com: the official Java website which documents all Java class API's
- http://spritedatabase.net/: a good source for freely available sprites to use in 2D games
- http://student.computing.dcu.ie/blogs/eoincos/: the blog for the development of the project
- http://www.totheriver.com/learn/xml/xmltutorial.html: a great little tutorial which helps bridge the gap between XML and Java
- com.sun.org.apache.xml: this Java package is used to output a DOM to an XML file

7 Installation Guide

After downloading and unzipping the game (which is freely available from the website http://fluffymachappy.com/Game/SwitchBall.zip), you are all set up. There is no installation involved once you have the necessary files downloaded. To run the game double-click SwitchBall.jar which is located within the folder SwitchBall. Requires Java 2 Standard Edition (J2SE) version 1.4 or later to run.

8 User Testing

User A:

Pro's:

- likes the intelligence of the AI characters.

Con's:

- did not notice the time limit on the bottom left. The time limit was moved to the bottom middle. Also added a countdown timer which beeps at second intervals at the final 5 seconds.
- finds the ball fumbling animation confusing. This is when the ball fumbles and immediately goes to the closest opposing player. This could be resolved by leaving the ball where it is fumbled and allowing users to pick up the ball.
- confused about what is happening during some parts of the game. This was resolved by adding a display box on the right hand side which shows current information about the state of the game.

User B:

Pro's:

- liked how the game is embedded onto the website and can be easily played in an internet browser.
- found the game very addictive to play.

Con's:

- did not know the controls of the game. This was resolved by adding in a controls page to the website page and to the offline game menu.
- did not notice the time limit on the bottom left. Resolved in User A testing.
- did not like some of the selectable maps as the colours were too overpowering. Added a new map called Clouds which seems to work well with the colours in the game.
- could not figure out which team some of the AI characters were playing on. This could be resolved by separate team colours by creating new images for each player with one colour red and another blue.

User C:

Con's:

- did not like the fancy background on the match results page. This was replaced by a plain orange background which matches the colour of the walls in the game.

User D:

Pro's:

- found the game very addictive to play.

Con's:

- user was not sure how far their player could pass.
- found that the players are not well balanced in their unique attributes. Found that a winged player could easily get around the slower players and score direct goals.
- wants a 4 player mode with extra goals on the left and right side. This would also require a larger field and network play to make it usable.

9 Testing

There are many units which were fully tested before integrating into the system. Below are the most important of these tests which clearly show the aim of this project. These tests focused on usability for the end user, component integration and manageability. As a result of the program being done in Java, many of the issues of cross platform have been resolved by the JVM. Once the user has the necessary version of Java installed on their Operating System, this program will run on the installed JVM.

Embedding on a Website

This feature allows two users to play a quick match of the game before actually downloading the game on the internet. This requires an internet connection and Java installed in the internet browser.

Browser	*Result*
Google Chrome V4.1	Correct
Mozilla Firefox V3.5.6	Correct
Internet Explorer V8.0	Correct

XML Output for Teams

This tests that the program correctly converts the list of teams to XML. We first convert the list of teams to an intermediary DOM which can then be used by the Apache XMLSerializer to properly output the XML.

List of Teams	*Result*
Null	Correct
Default list of teams	Correct
User added teams	Correct

Pass Ball

This tests to see if the method correctly sets the direction of the ball towards the selected sprite from the original sprite who has the ball.

Input Sprite	*Result*
Null	Correct, original sprite remains in control of ball
Sprite is in first quadrant	Correct
Sprite is in second quadrant	Correct
Sprite is in third quadrant	Correct
Sprite is in fourth quadrant	Correct
Sprite has a relative x of 0	Correct, no anomalies occur
Sprite has a relative y of 0	Correct

Ready Moods

This tests to make sure each mood is given the proper weight depending on the state. Each mood gets its weights from different parts of the state. Each weight is a number between 0 and 1.

Move Towards Goal

If his team does not have the ball, this weight is set to 0. Otherwise it increases the further the player gets from the opposing goal.

Opposing Player Nearby

If his team does not have the ball, this weight is set to 0. Otherwise it increases the closer the player gets to his closest opposing player.

Mark Player

If his team does have the ball, this weight is set to 0. Otherwise it increases the closer the player gets to his closest opposing player.

Mark Goal

If his team does have the ball, this weight is set to 0. Otherwise it increases the closer the player gets to the midway point between his team goal and his closest opposing player.

Mark Position

This weight increases the further the player gets away from his mark position.

Get Mark Position

This tests to make sure that each AI player knows the location they should be marking. This Y location is always the Y location of the captain. The X location is based on their overall position (left fielder, mid fielder or right fielder). This location is usually between the captain and the corresponding wall.

Position	*Captain's Position*	*Result*
Left fielder	Left fielder	Correct, this is the captain's location as both players have the same position
Mid fielder	Left fielder	Correct, this is closer to the captain than the wall on the right side
Right fielder	Left fielder	Correct, this is closer to the wall than the captain on the right side
Left fielder	Mid fielder	Correct, this is halfway between the captain and the wall on the left side
Mid fielder	Mid fielder	Correct, this is the captain's location
Right fielder	Mid fielder	Correct, this is halfway between the captain and the wall on the right side
Left fielder	Right fielder	Correct, this is closer to the wall on the left side
Mid fielder	Right fielder	Correct, this is closer to the captain on the right

		side
Right fielder	Right fielder	Correct, this is the captain's location

Move Based on Angle

This method allows us to get an angle based on a relative x and y and convert it to an action that a sprite can take. This action is a movement in a specific direction. As a result of GTGE coordinate system, we had to invert the y coordinates. This is to make (0, 0) coordinate start off as the bottom left instead of the top left as in GTGE.

Relative X	*Relative Y*	*Result*
0	0	Correct, defaults to move up
0	Positive number	Correct, move up
0	Negative number	Correct, move down
Positive number	0	Correct, move right
Negative number	0	Correct, move left
5	5	Correct, move up right
-5	5	Correct, move up left
-5	-5	Correct, move down left
5	-5	Correct, move down right
1	20	Correct, move up
-1	20	Correct, move up
1	-20	Correct, move down
-1	-20	Correct, move down
20	1	Correct, move right
20	-1	Correct, move right
-20	1	Correct, move left
-20	-1	Correct, move left

Disclaimer: A report submitted to Dublin City University, School of Computing for the 4th Year Project, 2009/2010. I hereby certify that the work presented and the material contained herein is my own except where explicitly stated references to other material are made.

DS1

SwitchBall

Sample Code

Eoin Costelloe (56586261)

4th Year Project – DCU May 2010

Supervisor : Mr David Sinclair

Introduction:

Below are several pieces of code which show aspects of my program that have greater interest. Please note that these functions have been fully tested and work properly. This should also illustrate that this program is made of many different modules which are effectively compartmentalized to increase manageability and follow the logical flow of the process.

Save Team (MenuFrame):

Here I convert the list of teams and their associated players to a Document Object Model (DOM) and then save this DOM as XML to a local file called teams.xml.

```
//convert team object into a DOM and save as XML
try
{
   //convert current list of teams into DOM
   DocumentBuilderFactory currentDocumentBuilderFactory =
DocumentBuilderFactory.newInstance();
   DocumentBuilder currentDocumentBuilder =
currentDocumentBuilderFactory.newDocumentBuilder();
   Document currentDocument = currentDocumentBuilder.newDocument();

   //create root element
   Element currentTeamsElement = currentDocument.createElement("teams");
   currentDocument.appendChild(currentTeamsElement);

   //go through each team whcih isnt default and add to DOM tree
   for(int i = 3; i < listOfTeams.size(); i++)
   {
      //define team element
      Element currentTeamElement = currentDocument.createElement("team");
      Team currentTeam = listOfTeams.get(i);
      currentTeamElement.setAttribute("name", currentTeam.getName());

      //define left player element
      Element currentLeftPlayerElement  = currentDocument.createElement("player");
      currentLeftPlayerElement.setAttribute("name",
currentTeam.getLeftPlayer().getName());
      currentTeamElement.appendChild(currentLeftPlayerElement);

      //define center player element
      Element currentCenterPlayerElement  = currentDocument.createElement("player");
      currentCenterPlayerElement.setAttribute("name",
currentTeam.getCenterPlayer().getName());
      currentTeamElement.appendChild(currentCenterPlayerElement);

      //define right player element
      Element currentRightPlayerElement  = currentDocument.createElement("player");
      currentRightPlayerElement.setAttribute("name",
currentTeam.getRightPlayer().getName());
      currentTeamElement.appendChild(currentRightPlayerElement);

      currentTeamsElement.appendChild(currentTeamElement);
   }

   //define the format for the outputting XML and output the file
   OutputFormat XMLFormat = new OutputFormat(currentDocument);
   XMLFormat.setIndenting(true);
   XMLFormat.setLineSeparator("\r\n");
   XMLSerializer serializer = new XMLSerializer(new FileOutputStream(new
File("teams.xml")), XMLFormat);
   serializer.serialize(currentDocument);


   JOptionPane.showMessageDialog(this, "Team data successfully saved.", "",
JOptionPane.INFORMATION_MESSAGE);
} catch(Exception e) {
```

```
   JOptionPane.showMessageDialog(this, "Problem saving Team data.", "",
JOptionPane.ERROR_MESSAGE);
}
```

Difficulty (SwitchBall):

This code defines the frequency that a player sorts their mood array. The more steps they take before they sort this array, the more likely they will miss higher-priority moods as they appear.

```
//the more steps an AI player goes before changing its mood
//means its easier to predict
int numberSteps;
if(difficulty == PlayerSprite.EASY)
{
   numberSteps = 100;
}
else if(difficulty == PlayerSprite.MEDIUM)
{
   numberSteps = 30;
}
else if(difficulty == PlayerSprite.HARD)
{
   numberSteps = 5;
}
else
{
   numberSteps = 30;
}
```

Pass Ball (PlayerSprite):

This method effectively creates the correct path between two players and sends the ball along this path at a small speed. This appears on-screen as the ball moving along the screen towards the other player.

```
void passBall(Sprite playerToPass)
{
   //if the player has the ball, they can pass
   if(currentBall != null)
   {
      //find the relative x and y direction of the ball based on origin
      double relativeXDirection = (playerToPass.getX() - getX());
      double relativeYDirection = (playerToPass.getY() - getY());
      //find the slope, (y2 - y1)/(x2 - x1)
      double slopeBetweenPlayers = (relativeYDirection / relativeXDirection);
      //try and move 0.2 in one axis and a smaller amount in the other axis
      //based on the equation y = x*slope and x = y/slope
      //if the slope is greater than 1 or less than -1, we move 0.2 in the y axis (and
find x)
      if((slopeBetweenPlayers > 1)||(slopeBetweenPlayers < -1))
      {
         if(relativeYDirection > 0)
         {
            currentBall.setSpeed((0.2/slopeBetweenPlayers), 0.2);
         }
         else
         {
            currentBall.setSpeed((-0.2/slopeBetweenPlayers), -0.2);
         }
      }
      //if the slope is between 1 and -1, we move 0.2 in the x axis (and find y)
      else if((slopeBetweenPlayers <= 1)&&(slopeBetweenPlayers >= -1))
      {
         if(relativeXDirection > 0)
         {
            currentBall.setSpeed(0.2, (0.2*slopeBetweenPlayers));
         }
         else
```

```
         {
            currentBall.setSpeed(-0.2, (-0.2*slopeBetweenPlayers));
         }
      }

      //do not catch the ball if you threw it
      currentBall.lostPossesion(this);

      //lose the ball
      currentBall = null;
   }
}
```

Ready Moods (PlayerSprite):

This makes sure the moods are given proper priority depending on whether they are defending or attacking. Marking your position is common in both and gives the user slight control of the AI positions.

```
private void readyMoods(PlayerSprite yourTeamCaptain, PlayerSprite closestOpposingPlayer,
GoalSprite yourTeamGoal, GoalSprite opposingTeamGoal)
{
   //if your team has the ball, be offensive
   if(yourTeamCaptain.hasBall())
   {
      //set moveTowardsGoal weight
      moveTowardsGoal.setWeight(moveTowardsGoalWeight(opposingTeamGoal));
      //set opposingPlayerNearby weight
      opposingPlayerNearby.setWeight(opposingPlayerNearbyWeight(closestOpposingPlayer));
      markPlayer.setWeight(0);
      markGoal.setWeight(0);
      markPosition.setWeight(markPositionWeight(yourTeamCaptain));
   }
   //else be defensive
   else
   {
      moveTowardsGoal.setWeight(0);
      opposingPlayerNearby.setWeight(0);
      //set markPlayer weight
      markPlayer.setWeight(markPlayerWeight(closestOpposingPlayer));
      //set markGoal weight
      markGoal.setWeight(markGoalWeight(yourTeamGoal, closestOpposingPlayer));
      markPosition.setWeight(markPositionWeight(yourTeamCaptain));
   }
}
```

Distance Between Two Points:

This is a very important method in any trigonometric system. This is used for calculating the distance between two sprites using their x and y coordinates.

```
//distance between two points = square root(squared(x2 - x1) + squared(y2 - y1))
static double distanceBetweenTwoPoints(double point1X, double point1Y, double point2X,
double point2Y)
{
   return Math.sqrt(Math.pow((point1X - point2X), 2) + Math.pow((point1Y - point2Y), 2));
}
```

Get Mark Position X (PlayerSprite):

This is used to find the proper dynamic location that each player should be marking. The captain in this case is the user. Their position and location should define where each AI player should be located. This takes into account the walls of the field.

```
//try and find your position based on the captains position
//for example if captain is mid fielder and you are left fielder
```

```
//find the location halfway between the captain and the wall to his left side
double getMarkPositionX(double captainX, double captainPosition, double currentPosition)
{
   //left fielder
   if(currentPosition == 0)
   {
      //captian is mid fielder
      if(captainPosition == 1)
         return ((captainX - 50) / 2) + 50;
      //captian is right fielder
      else if(captainPosition == 2)
         return ((captainX - 50) / 3) + 50;
      else
         return captainX;
   }
   //mid fielder
   else if(currentPosition == 1)
   {
      //captian is left fielder
      if(captainPosition == 0)
         return (((SwitchBall.WIDTH - 50) - captainX) / 3) + captainX;
      //captian is right fielder
      else if(captainPosition == 2)
         return (((captainX - 50) / 3) * 2) + 50;
      else
         return captainX;
   }
   //right fielder
   else //currentPosition == 2
   {
      //captian is left fielder
      if(captainPosition == 0)
         return ((((SwitchBall.WIDTH - 50) - captainX) / 3) *2) + captainX;
      //captian is mid fielder
      else if(captainPosition == 1)
         return (((SwitchBall.WIDTH - 50) - captainX) / 2) + captainX;
      else
         return captainX;
   }
}
```

Find Closest Player (PlayerSprite):

This method is used by the AI players to generate their state. The AI state contains the AI player's team captain, the closest opposing player, the AI player's team goal and the opposing team goal. This method does a simple search through the opposing team players to find the closest one to the AI player.

```
public static PlayerSprite findClosestPlayer(Sprite inputPlayer, ArrayList<PlayerSprite>
inputTeam)
{
   //default is the first opposing player
   double closestOpposingPlayerDistance =
distanceBetweenTwoPoints(inputTeam.get(0).getX(), inputTeam.get(0).getY(),
inputPlayer.getX(), inputPlayer.getY());
   int closestOpposingPlayerIndex = 0;

   double currentOpposingPlayerDistance;

   for(int currentOpposingPlayerIndex = 1; currentOpposingPlayerIndex < inputTeam.size();
currentOpposingPlayerIndex++)
   {
      currentOpposingPlayerDistance =
distanceBetweenTwoPoints(inputTeam.get(currentOpposingPlayerIndex).getX(),
inputTeam.get(currentOpposingPlayerIndex).getY(), inputPlayer.getX(),
inputPlayer.getY());
      if(currentOpposingPlayerDistance < closestOpposingPlayerDistance)
      {
         closestOpposingPlayerDistance = currentOpposingPlayerDistance;
```

```
            closestOpposingPlayerIndex = currentOpposingPlayerIndex;
        }
    }

    return inputTeam.get(closestOpposingPlayerIndex);
}
```

Move Based on Angle (PlayerSprite):

We use this method to define what is the correct action to take based on the relative x and y. We use these relative directions to generate the angle and so find the quadrants. These quadrants allow us to select the appropriate action.

```
private int moveBasedOnAngle(double relativeXDirection, double relativeYDirection)
{
    //if x is zero, move based on y only
    double angle = 1;

    //change y direction from + downwards (standard in games) to + upwards (standard in
maths)
    relativeYDirection = -relativeYDirection;

    if(relativeXDirection != 0)
    {
        //get angle between sprites
        angle = Math.atan(relativeYDirection / relativeXDirection);

        //normalise angle which is currently in the range pi/2 and -pi/2
        angle = (angle / Math.PI) * 2;
    }

    if(relativeYDirection >= 0)
    {
        if(angle == 0)
        {
            if(relativeXDirection >= 0)
            {
                return MOVERIGHT;
            }
            else
            {
                return MOVELEFT;
            }
        }
        if((angle > 0)&&(angle < 0.25))
        {
            return MOVERIGHT;
        }
        else if((angle >= 0.25)&&(angle < 0.75))
        {
            return MOVEUPRIGHT;
        }
        else if((angle < 0)&&(angle >= -0.25))
        {
            return MOVELEFT;
        }
        else if((angle < -0.25)&&(angle >= -0.75))
        {
            return MOVEUPLEFT;
        }
        else //if((angle >= 0.75)||(angle < -0.75))
        {
            return MOVEUP;
        }
    }
    else
    {
        if(angle == 0)
        {
```

```
            if(relativeXDirection >= 0)
            {
               return MOVERIGHT;
            }
            else
            {
               return MOVELEFT;
            }
         }
         if((angle > 0)&&(angle < 0.25))
         {
            return MOVELEFT;
         }
         else if((angle >= 0.25)&&(angle < 0.75))
         {
            return MOVEDOWNLEFT;
         }
         else if((angle < 0)&&(angle >= -0.25))
         {
            return MOVERIGHT;
         }
         else if((angle < -0.25)&&(angle >= -0.75))
         {
            return MOVEDOWNRIGHT;
         }
         else //if((angle >= 0.75)||(angle < -0.75))
         {
            return MOVEDOWN;
         }
      }
   }
```