UNIVERSITY OF HERTFORDSHIRE
School of Computer Science

BSc Honours in Computer Science

6COM1056 - Software Engineering Project

Final Report
April 2019

Ice Hockey Simulation Engine

L.Oram

Supervised by: Olenka Marczyk

1. Introduction

1.1. Project Introduction

This project aims to design, implement, test, and evaluate a simulation engine for a game of ice hockey. The planned system will simulate two teams of players with a number of attributes facing off against each other. This project will discuss the decisions taken regarding the technologies used, the implementation methods, and the testing process.

1.2. Project Motivation

Being an ice hockey fan. This project provides me the opportunity to break the sport down into basic components and think about how they are linked, along with factors not considered when just watching a game.

Another reason is that I have used and still use similar simulation engine systems in the past. While using these systems, which will be discussed in further detail later in this project, I have always thought about improvements and changes that could be implemented. The chance to design my own system that could contain some of these features was a major contributing factor to the motivation for this project.

A more technical reason for this project is that I wanted to choose a project that would challenge me and allow me to explore artificial intelligence and graphical implementation of software along with continuing to develop my java programming skills.

1.3. Aims and Objectives

This project can be split into a number of aims and objectives that will guide the design of the system. Some of these aims and objectives will reference the rules of ice hockey, a simple explanation of these can be found in the appendix.

1.3.1. Aims

- Create a simulation engine that produces a realistic result of an ice hockey game between two teams.
- Display a graphical representation of the gameplay as it happens along with team specific player icons and correct player numbers displayed.
- Learn about and develop an AI that produces realistic results.
- Develop my knowledge of implementing a larger complex system with java containing a graphical component.

1.3.2. Objectives

- Import player data from a file containing all player details and attributes
- Start up a game with two teams of players in their correct positions on the ice with players being displayed on screen in their correct team colours with their correct number displayed clearly.
- Puck and players will have full collision with the puck bouncing off the boards of the arena.
- Begin the game with an opening faceoff where the puck will be passed to a player
- Continue to simulate a complete game of ice hockey
- A player will be able to travel to a loose puck and pick it up.
- The player will then be able to skate with the puck, pass the puck, or shoot the puck at the net.
- Players on the same team as the player with the puck will attempt to find open ice and move to a position that is accurate to their position and beneficial to the team.
- Players not on the same team will attempt to block passing lanes, shooting lanes, mark other players, and attempt to challenge for the puck or check (hit) the player with the puck.
- Player decisions will be influenced by a number of factors including their attributes, position on ice and position of other players on ice.
- Success rates of actions and accuracy of passes and shots will be influenced by the players attributes.
- Goalies will be able to save shots on goal at a realistic rate and based off of both the player shooting attributes and the goalie attributes.
- Rate of player passing vs scoring and success rates of these actions will be reflective of real live statistics.
- Total Goals and shots on goal for each team will be tracked and displayed
- Player actions and stats will be stored and available to be displayed.

1.4. Report Structure

Chapter 2 will discuss a number of existing products that can be used to simulate an ice hockey game. This report will discuss the methods they use, their strengths and weaknesses as a simulation engine for ice hockey, and what features will influence this project.

Chapter 3 of this report breaks down the design choices made when planning this project, including initial code and database architecture. Decision made that influence the choices made on how ideas will be implemented are also explained here.

Chapter 4 goes into detail on the implementation of the system. The technologies I decided to use will be set out here with the reasoning behind my choices. This chapter will then continue to explain the methods used to a implement a number of features in detail.
Chapter 5 explains the testing methods used to ensure the final system was working to an acceptable level. This chapter also discusses the difficulties faced during testing and the issues surrounding testing a graphical system with a large number of random components.

Chapter 6 provides a final evaluation of the system, the success and failures of the system, and changes I would have like to make to the final system. This chapter will provide a reflection back on the project as a whole.

2. Research

2.1. Existing Products

2.1.1. EA NHL

The EA series of NHL games are the most widely known examples of a simulator engine, not only for ice hockey but across a range of sports. EA are a large company with a big history of creating sports games and their NHL series is no exception. Of the existing products here EA NHL has by far the most development and budget.

The focus of this product however is the gameplay rather than the actual AI and simulation aspects which are lacking somewhat compared to other titles. This isn’t to say that it is a bad simulation with results being relatively realistic.

When simulating games the user has the option to either watch the game in full or to receive a final box score. The graphics of the game are full 3d high quality renders. The physics system in the game can on occasion be less than ideal.

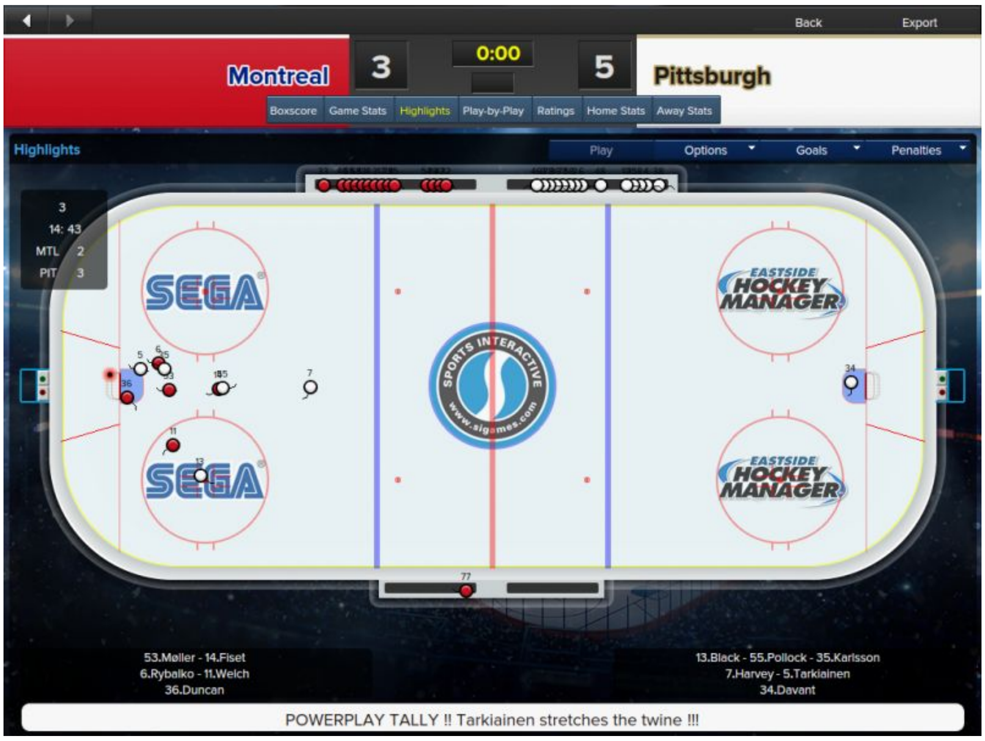
While EA NHL offers the best visual representation of a hockey game, the actual AI and simulation engine leaves something to be desired. It’s focus on player controlled gameplay rather than simulating two AI controlled teams also detracts from its suitability for the requirements of this project.

The user interface of the game is relatively simple and intuitive, especially when compared to the other products mentioned here. A simple to use interface is something I would like to aim for in this project. The main methods used to make the interface as simple as possible to use is to simplify and cut down on what is available for the user.

2.1.2. Eastside Hockey Manager and Franchise Hockey Manager

Eastside and Franchise Hockey Manager (ESHM and FHM) are two similar games that aim to replicate the ice hockey experience from a general managers point of view. Due to their similarities I will be discussing both games together.

The graphical representation of the simulation in these two games is a rather basic 2D top down view as can be seen in the image below. However, it does a great job at displaying the important information to the user. It is easy to understand and does not have and fancy graphics. This type of graphical representation is the best method for a pure simulation system. This is the format I will be aiming for in my system.



53.Møller - 14.Fiset
6.Rybalko - 11.Welch
36.Duncan

13.Black - 55.Pollock - 35.Karlsson
7.Harvey - 5.Tarkiainen
34.Davant

POWERPLAY TALLY !! Tarkiainen stretches the twine !!!

The simulation of these two games is incredibly complex. Players in each game have roughly 30 different attributes that affect their behaviour in game. The result of this number of variables to a player is that each player behaves very differently to each other. However the vast number of attributes can overwhelm players with the amount of data. The overall simulation in these games is relatively realistic.

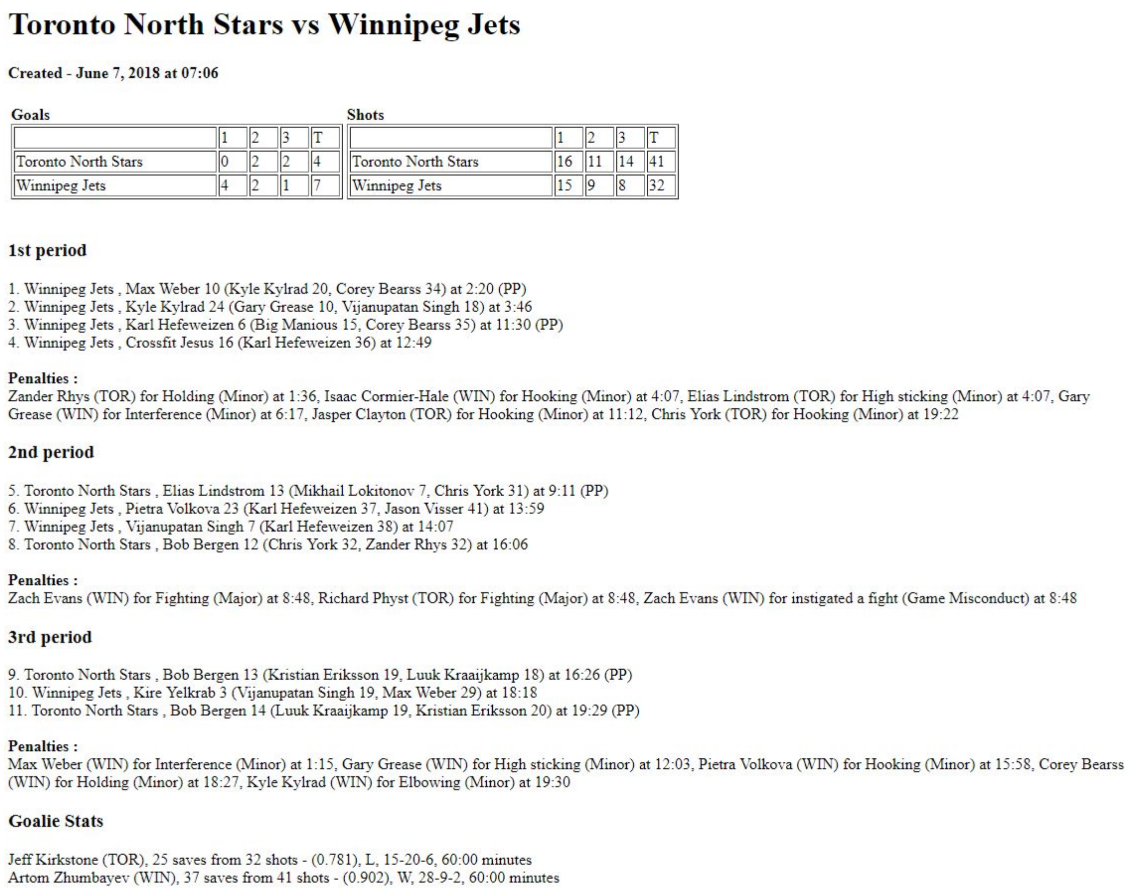
2.1.3. Simon T Hockey Simulator

The Simon T Hockey Simulator (STHS) is the existing product I have the most experience with and a large part of the motivation behind this project. Unlike the two previous products, STHS has been created purely as a simulation engine rather than as a commercial game. This focus on the simulation leads to it being the most realistic of the products mentioned here.

There are a number of different simulation engine versions through the STHS system, all of which behave slightly differently and have pros and cons depending on the intended use case. Unless stated otherwise this report will focus on version 1.5 of STHS which produces the most realistic scores of all versions currently available.

The user interface of STHS can be split into two components, the management or set up side and the actual game output. The management side of STHS consists of two separate programs. The first program is the “client”, this is used by managers of the teams to set lines and strategies for the simulation. The second program is the full simulator, this is where teams can be created and edited, fixtures created, and where game simulations are run from.

The game output of STHS consists of a basic text only HTML page as shown in the image below. As you can see compared to the other products discussed this is by far the simplest. It is also the only product to not have the option to watch a game play out live.



Toronto North Stars vs Winnipeg Jets

Created - June 7, 2018 at 07:06

Goals

	1	2	3	T
Toronto North Stars	0	2	2	4
Winnipeg Jets	4	2	1	7

Shots

	1	2	3	T
Toronto North Stars	16	11	14	41
Winnipeg Jets	15	9	8	32

1st period

1. Winnipeg Jets , Max Weber 10 (Kyle Kylrad 20, Corey Bearss 34) at 2:20 (PP)
2. Winnipeg Jets , Kyle Kylrad 24 (Gary Grease 10, Vijanupatan Singh 18) at 3:46
3. Winnipeg Jets , Karl Hefeweizen 6 (Big Manious 15, Corey Bearss 35) at 11:30 (PP)
4. Winnipeg Jets , Crossfit Jesus 16 (Karl Hefeweizen 36) at 12:49

Penalties :
Zander Rhys (TOR) for Holding (Minor) at 1:36, Isaac Cormier-Hale (WIN) for Hooking (Minor) at 4:07, Elias Lindstrom (TOR) for High sticking (Minor) at 4:07, Gary Grease (WIN) for Interference (Minor) at 6:17, Jasper Clayton (TOR) for Hooking (Minor) at 11:12, Chris York (TOR) for Hooking (Minor) at 19:22

2nd period

5. Toronto North Stars , Elias Lindstrom 13 (Mikhail Lokitonov 7, Chris York 31) at 9:11 (PP)
6. Winnipeg Jets , Pietra Volkova 23 (Karl Hefeweizen 37, Jason Visser 41) at 13:59
7. Winnipeg Jets , Vijanupatan Singh 7 (Karl Hefeweizen 38) at 14:07
8. Toronto North Stars , Bob Bergen 12 (Chris York 32, Zander Rhys 32) at 16:06

Penalties :
Zach Evans (WIN) for Fighting (Major) at 8:48, Richard Physt (TOR) for Fighting (Major) at 8:48, Zach Evans (WIN) for instigated a fight (Game Misconduct) at 8:48

3rd period

9. Toronto North Stars , Bob Bergen 13 (Kristian Eriksson 19, Luuk Kraaijkamp 18) at 16:26 (PP)
10. Winnipeg Jets , Kire Yelkrab 3 (Vijanupatan Singh 19, Max Weber 29) at 18:18
11. Toronto North Stars , Bob Bergen 14 (Luuk Kraaijkamp 19, Kristian Eriksson 20) at 19:29 (PP)

Penalties :
Max Weber (WIN) for Interference (Minor) at 1:15, Gary Grease (WIN) for High sticking (Minor) at 12:03, Pietra Volkova (WIN) for Hooking (Minor) at 15:58, Corey Bearss (WIN) for Holding (Minor) at 18:27, Kyle Kylrad (WIN) for Elbowing (Minor) at 19:30

Goalie Stats

Jeff Kirkstone (TOR), 25 saves from 32 shots - (0.781), L, 15-20-6, 60:00 minutes
Artom Zhumbayev (WIN), 37 saves from 41 shots - (0.902), W, 28-9-2, 60:00 minutes

One issue that STHS has is that it is a windows application only, while there is a mac application available this is extremely limited in functionality and rarely works in the first place. Making this system cross-platform compatible is a big aim for this project.

The player attribute system is the system I have decided to use for this project. Players have 13 attributes and goalies have 11 attributes, A full list of attributes and a brief description of how they influence the simulation can be found in Appendix C. I find this number of attributes allows the system and player creation to remain relatively simple while still allowing customisation of each player and having enough attributes to influence the simulation.

3. Design

3.1. Design Choices

3.1.1. Player importing

The first design decision was to decide on how I would store and import data including players, their attributes, and teams lines and strategies. The player data I will be using for this system comes from an existing file I have used when using the Simon T Hockey Simulator. The player file contains player details such as name, country, position (in the form of a number representation) and then the players attributes. A part of the file used can be found in appendix B.

The initial idea was to import players from this large CSV file. This would be the same method that STHS uses for player import. As for team lines and strategies Simon T uses its own file type (.stc,.sth,.shl). For my system I initially decided to attempt to also use CSV files for this data. At this point I began to create a prototype system that imported from a file and printed the output in an ordered fashion. The creation of this small program showed that CSV parsing was not the simplest solution to implement along with a number of problems that it would bring in the full system. One of the main issues over than implementation I had with the CSV file was that I would have to import the entire file of all teams at startup.

Switching to a database to store this data allowed me to access data much more simply through SQL queries vs the CSV parser I initially implemented. The use of SQL queries has allowed me to only import the players I want for the teams that are playing in the game rather than having to read an entire file at startup.

3.1.2. Graphical Representation

Players would be represented on screen as small circles with their player number displayed in the center of the circle. Each team would have their own colours for both home and away games. The initial idea for the implementation of this was to dynamically create the circles and text. However after initial attempts to implement this it was proving more of a challenge to implement than expected.

The second implementation test would be the implementation method used in the final product. Sprite Sheets were created containing all teams coloured circles with another sheet containing the teams numbers 0-9. When a player is created the system will get the correct circle and number or numbers to draw over the circle. This method proved much simpler to implement with the prototype system working flawlessly.

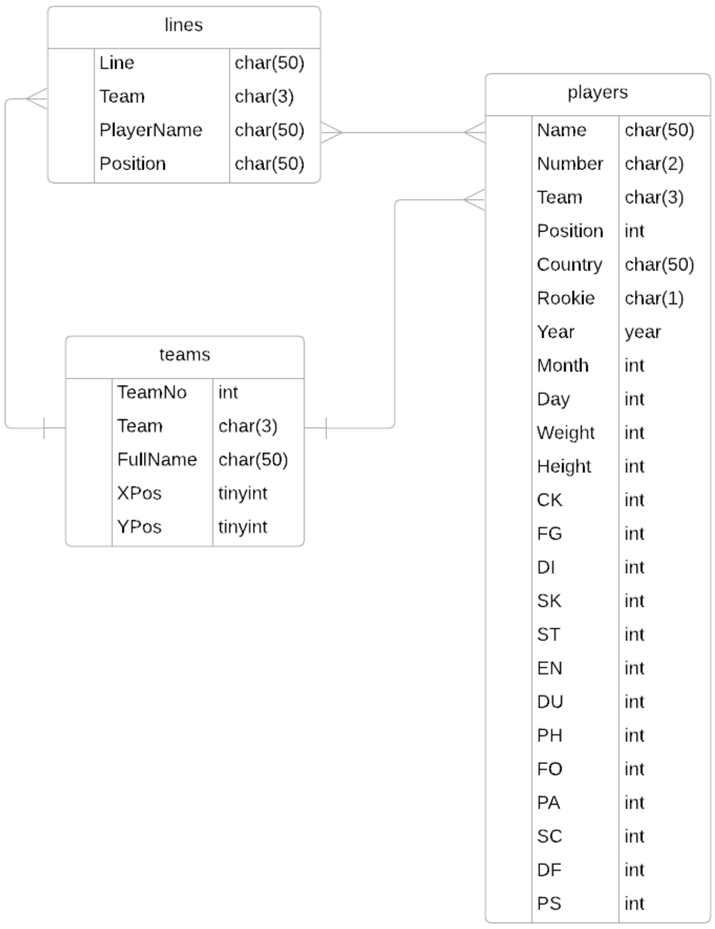
Home and away player and number spritesheets
(A number of teams have white numbering on their home colours)

3.1.3. Scenarios

As I began implementation of the system it became clear that I would need a way to clearly show that certain features of the simulation were working correctly. To solve this problem I implemented a variable in the code to turn off the randomness, I also created a special set of players with attributes that with no randomness would demonstrate certain features. The final addition made to the system was the scenarios. These are a small number of options to start a game with a different number of players and with the players and puck in a certain starting position so that they would act in a controlled way to display each aspect of the simulation.

3.2. Database Design

The system contains a relatively simple database. The database contains three tables, Players, Teams, and Lines. The player table contains all player information such as name, the team they play for, and their attributes. The team table contains team name, abbreviation and their position on the spritesheet. The Lines table contains the information needed to set the players on the ice for a game. Team will have a number of lines stored in this table. The playing position of a player on a line may be different to their natural position so this has to be stored.


lines

Line	char(50)
Team	char(3)
PlayerName	char(50)
Position	char(50)

teams

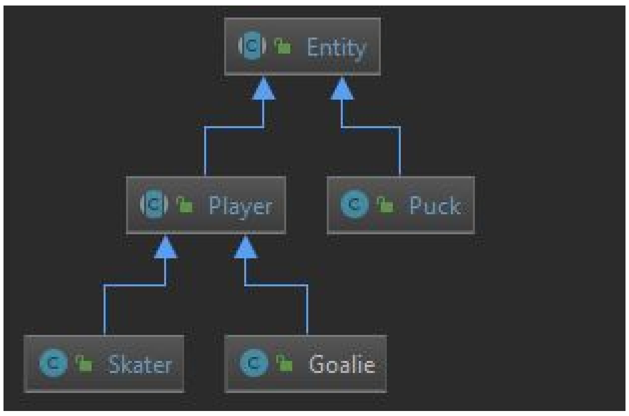
TeamNo	int
Team	char(3)
FullName	char(50)
XPos	tinyint
YPos	tinyint

players

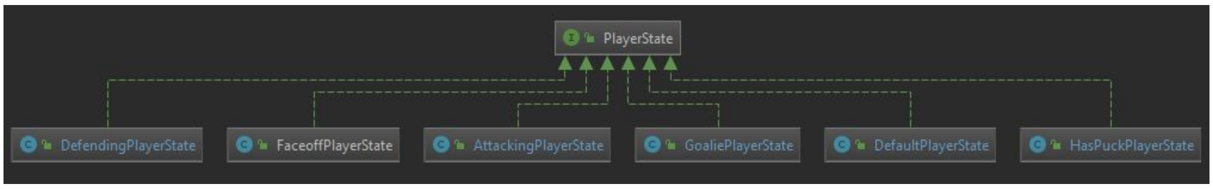
Name	char(50)
Number	char(2)
Team	char(3)
Position	int
Country	char(50)
Rookie	char(1)
Year	year
Month	int
Day	int
Weight	int
Height	int
CK	int
FG	int
DI	int
SK	int
ST	int
EN	int
DU	int
PH	int
FO	int
PA	int
SC	int
DF	int
PS	int

3.3. Code Design

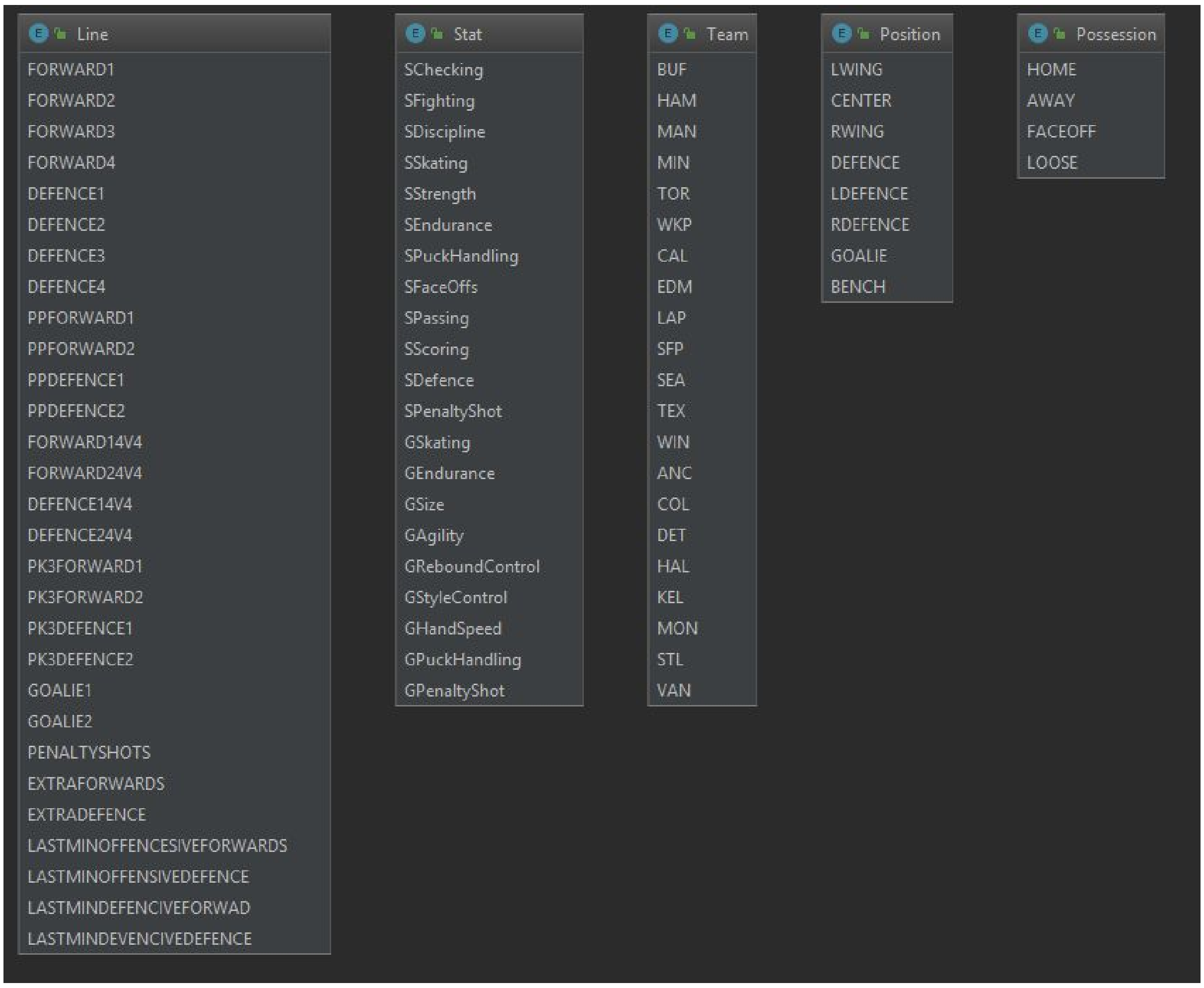
When designing the code the project was split into multiple smaller sections, the first section was the players and the puck. All of which will be displayed to the screen and will be moving about. Therefore they can inherit from a parent class “Entity” which will contain a number of fields and methods to handle the movement of these objects. Players can then further be split into Skaters and Goalies. The Player class will contain player details and a number of methods such as passing that both goalies and skaters will need.



Next we have the player state interface. This is designed around the state design pattern and will be used to control the players behaviours.



I will be using a number of enum classes for this system which can be seen below. These will allow me to easily compare many aspects of players and will be used to control the player states.



Line
- FORWARD1
- FORWARD2
- FORWARD3
- FORWARD4
- DEFENCE1
- DEFENCE2
- DEFENCE3
- DEFENCE4
- PPFORWARD1
- PPFORWARD2
- PPDEFENCE1
- PPDEFENCE2
- FORWARD14V4
- FORWARD24V4
- DEFENCE14V4
- DEFENCE24V4
- PK3FORWARD1
- PK3FORWARD2
- PK3DEFENCE1
- PK3DEFENCE2
- GOALIE1
- GOALIE2
- PENALTYSHOTS
- EXTRAFORWARDS
- EXTRADEFENCE
- LASTMINOFFENCESIVEFORWARDS
- LASTMINOFFENSIVEDEFENCE
- LASTMINDEFENCIVEFORWAD
- LASTMINDEVENCIVEDEFENCE

Stat
- SChecking
- SFighting
- SDiscipline
- SSkating
- SStrength
- SEndurance
- SPuckHandling
- SFaceOffs
- SPassing
- SScoring
- SDefence
- SPenaltyShot
- GSkating
- GEndurance
- GSize
- GAgility
- GReboundControl
- GStyleControl
- GHandSpeed
- GPuckHandling
- GPenaltyShot

Team
- BUF
- HAM
- MAN
- MIN
- TOR
- WKP
- CAL
- EDM
- LAP
- SFP
- SEA
- TEX
- WIN
- ANC
- COL
- DET
- HAL
- KEL
- MON
- STL
- VAN

Position
- LWING
- CENTER
- RWING
- DEFENCE
- LDEFENCE
- RDEFENCE
- GOALIE
- BENCH

Possession
- HOME
- AWAY
- FACEOFF
- LOOSE

The final full UML diagram can be seen in Appendix D (Attached separately) which shows the scale of this project.

4. Implementation

4.1. Chosen Platforms

4.1.1. Java

I decided to use Java for the code in this project. The main reason I picked java is because of familiarity with it after using it for the majority of my course. With the limited time to complete this project it would not have been feasible to look into learning another language that may be more suitable. However, java is a good language for this project and fulfills the needs of the system.

Using Java for this project brings many benefits such as it’s cross platform compatibility which allows the system to run on windows, linux and macOS. Another major feature of java is it’s suitability for object oriented programming as this system will rely on the principles of OOP heavily.

4.1.2. MySQL

For the database used in the system MySQL will be used. The reasoning behind the use of MySQL over other relational database management systems is its popularity and the amount of detailed documentation available. I have had some experience using MySQL in the past along with other RDMS.

4.1.3. Tools Used

My chosen IDE for this project was IntelliJ by jetbrains, I had considered using netbeans or eclipse for this project. However I decided on IntelliJ as I have used it briefly before along with netbeans which I have used over the duration of my course. I felt this project would be a good opportunity to learn IntelliJ more to bring it up to the level I’ve been at with netbeans.

When working with the database portion of the system I used HeidiSQL as my MySQL graphical client. Before this project I had minimal experience with MySQL clients however HeidiSQL was highly recommended and well reviewed.

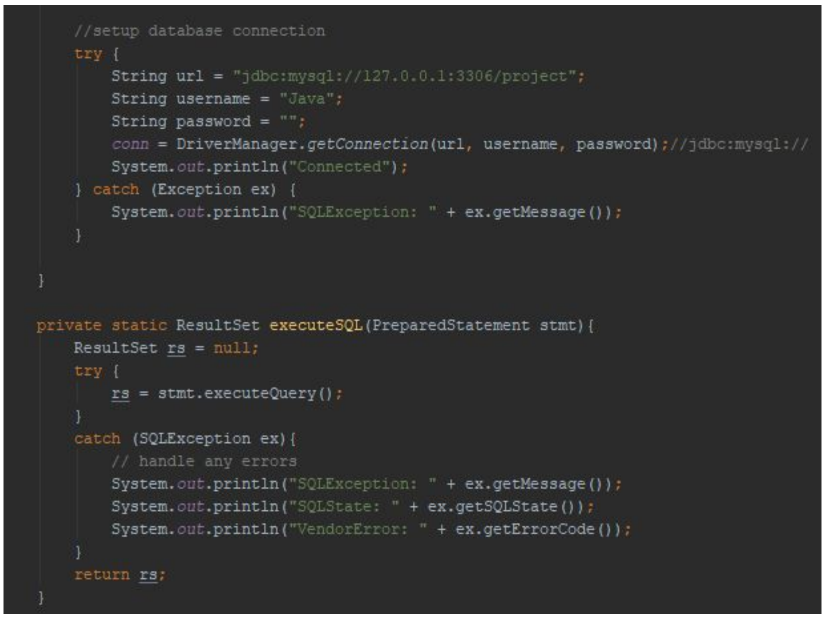
For the creation of the graphical elements of the system Paint.NET was used. The graphics used were simple circle representation and 8 bit display style numbers so a more advanced program such as photoshop would not add much to the project.

4.2. Code Implementation

Full code for this project can be viewed in Appendix E (attached Separately). This chapter will break down the key features of the system and the methods used to implement them.

4.2.1. Player Creation

Once a game is created the first step is to populate the teams with players. The players and all their details are stored in the database. The system has a single classes dedicated to accessing the database. The class contains separate methods for connection to the database and executing queries. The image below shows the two methods responsible for this.

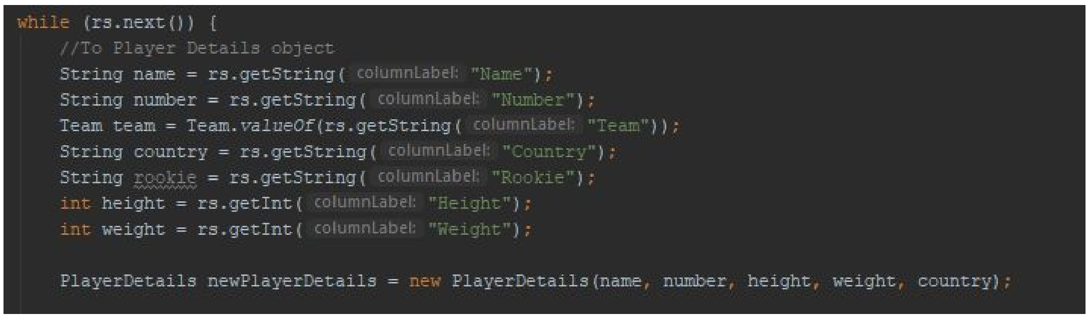


```java
        //setup database connection
        try {
            String url = "jdbc:mysql://127.0.0.1:3306/project";
            String username = "Java";
            String password = "";
            conn = DriverManager.getConnection(url, username, password);//jdbc:mysql://
            System.out.println("Connected");
        } catch (Exception ex) {
            System.out.println("SQLException: " + ex.getMessage());
        }

    }

    private static ResultSet executeSQL(PreparedStatement stmt){
        ResultSet rs = null;
        try {
            rs = stmt.executeQuery();
        }
        catch (SQLException ex){
            // handle any errors
            System.out.println("SQLException: " + ex.getMessage());
            System.out.println("SQLState: " + ex.getSQLState());
            System.out.println("VendorError: " + ex.getErrorCode());
        }
        return rs;
    }
```

The class also contains a number of other methods which handle the reading and parsing of the data. These methods create preparedStatements which are passed to the executeSQL method. These are then executed and a ResultSet containing all the data is returned. The method “loadTeam” creates a hashmap of all the players on a team.

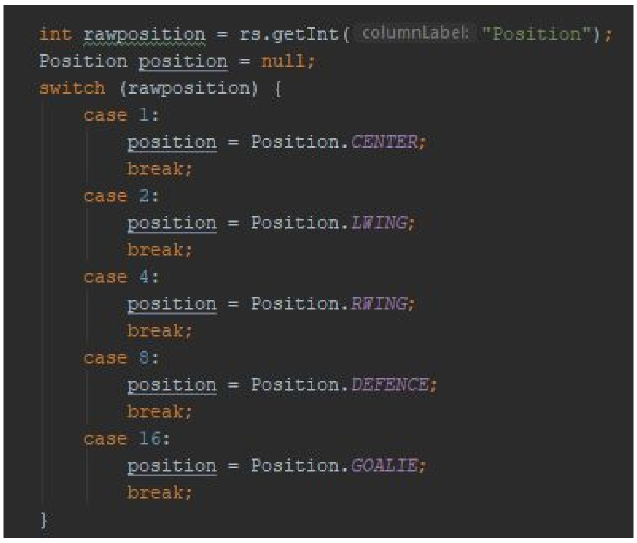
When the database is queried a resultset of all players on the team is returned with all their details and attributes. This result set is iterated over player by player and the player object is created and added to the hashmap with the players name being the key.

During the player creation loop the first step is to create a PlayerDetails object which contains all the personal information of a player. I chose to separate out player details into a seperate class to reduce the number of variables that would be passed to the player constructor.



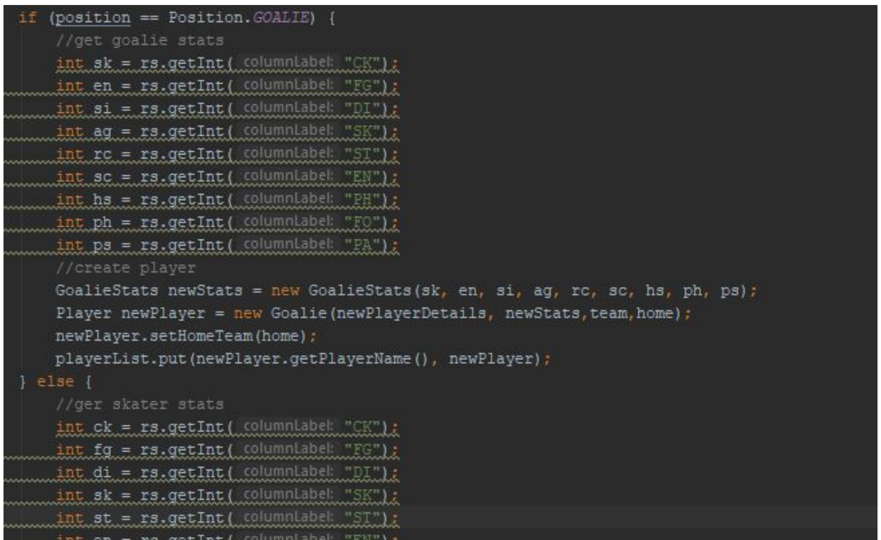
```java
while (rs.next()) {
    //To Player Details object
    String name = rs.getString( columnLabel: "Name");
    String number = rs.getString( columnLabel: "Number");
    Team team = Team.valueOf(rs.getString( columnLabel: "Team"));
    String country = rs.getString( columnLabel: "Country");
    String rookie = rs.getString( columnLabel: "Rookie");
    int height = rs.getInt( columnLabel: "Height");
    int weight = rs.getInt( columnLabel: "Weight");

    PlayerDetails newPlayerDetails = new PlayerDetails(name, number, height, weight, country);
```

Next goalies are separated from skaters as they have seperate classes that both inherit from the Player class. As the position information is stored as an integer I use a switch statement to assign a enum type to represent position rather than the integer as this simplifies future code in the system.



```java
int rawposition = rs.getInt( columnLabel: "Position");
Position position = null;
switch (rawposition) {
    case 1:
        position = Position.CENTER;
        break;
    case 2:
        position = Position.LWING;
        break;
    case 4:
        position = Position.RWING;
        break;
    case 8:
        position = Position.DEFENCE;
        break;
    case 16:
        position = Position.GOALIE;
        break;
}
```

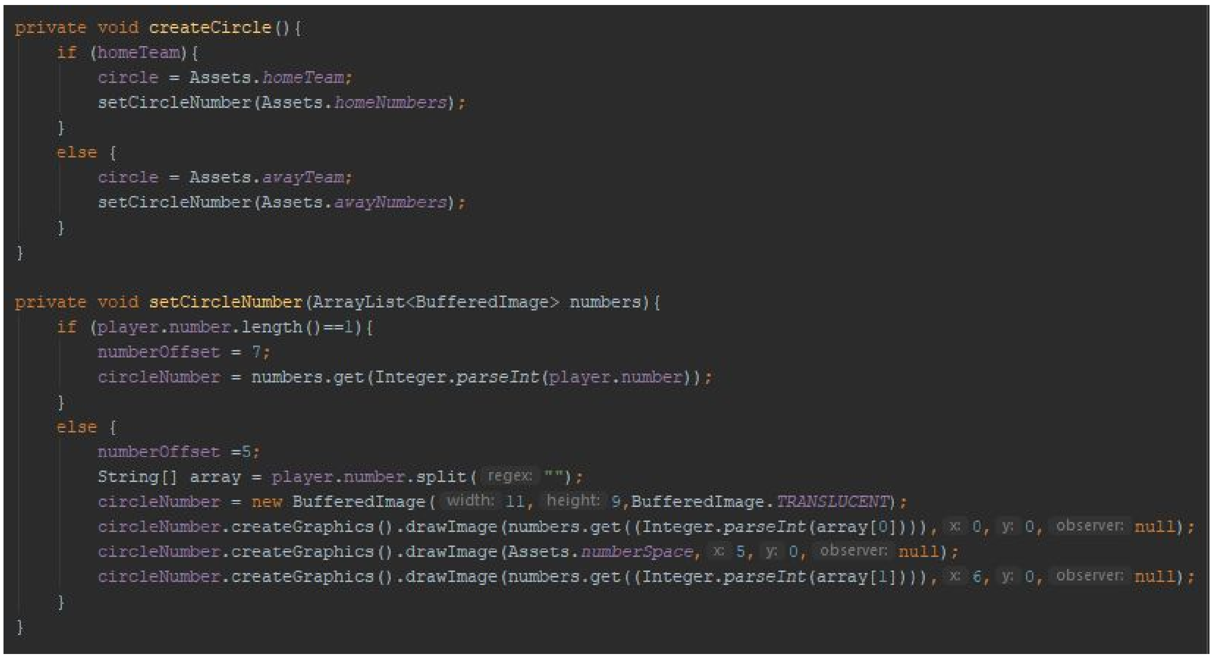
A Stats object is then created. This is either “SkaterStats” or “GoalieStats”. Again this is a separate class to reduce what is passed to the Skater or Goalie class.
 The next step is to create the actual Skater or Player object. “PlayerDetails” and the stats are passed to this along with the team the player is on and if they are on the home team or not.



```java
if (position == Position.GOALIE) {
    //get goalie stats
    int sk = rs.getInt( columnLabel: "CK");
    int en = rs.getInt( columnLabel: "FG");
    int si = rs.getInt( columnLabel: "DI");
    int ag = rs.getInt( columnLabel: "SK");
    int rc = rs.getInt( columnLabel: "ST");
    int sc = rs.getInt( columnLabel: "EN");
    int hs = rs.getInt( columnLabel: "PH");
    int ph = rs.getInt( columnLabel: "FO");
    int ps = rs.getInt( columnLabel: "PA");
    //create player
    GoalieStats newStats = new GoalieStats(sk, en, si, ag, rc, sc, hs, ph, ps);
    Player newPlayer = new Goalie(newPlayerDetails, newStats,team,home);
    newPlayer.setHomeTeam(home);
    playerList.put(newPlayer.getPlayerName(), newPlayer);
} else {
    //ger skater stats
    int ck = rs.getInt( columnLabel: "CK");
    int fg = rs.getInt( columnLabel: "FG");
    int di = rs.getInt( columnLabel: "DI");
    int sk = rs.getInt( columnLabel: "SK");
    int st = rs.getInt( columnLabel: "ST");
```

4.2.2. GUI

When a new player object is instantiated the players graphical representation is created. Both the circle used to represent the player and the numbers that are overlaid on the circle are stored as buffered images in the player class. Using the players team and if they are the home team the correct circle is cropped from the spritesheet.



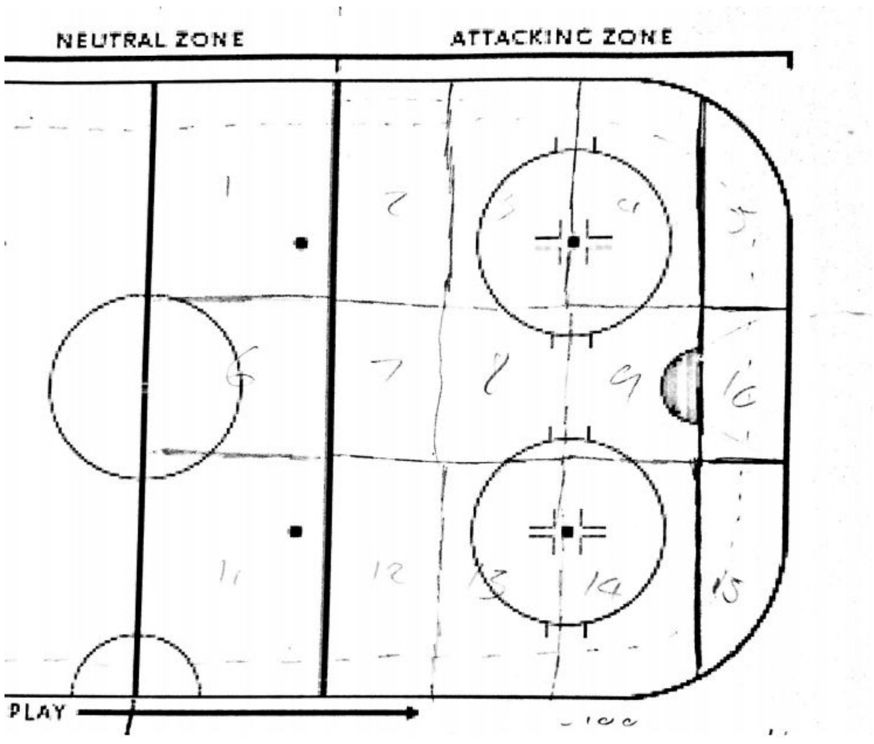
```
private void createCircle(){
    if (homeTeam){
        circle = Assets.homeTeam;
        setCircleNumber(Assets.homeNumbers);
    }
    else {
        circle = Assets.awayTeam;
        setCircleNumber(Assets.awayNumbers);
    }
}

private void setCircleNumber(ArrayList<BufferedImage> numbers){
    if (player.number.length()==1){
        numberOffset = 7;
        circleNumber = numbers.get(Integer.parseInt(player.number));
    }
    else {
        numberOffset =5;
        String[] array = player.number.split( regex: "");
        circleNumber = new BufferedImage( width: 11, height: 9,BufferedImage.TRANSLUCENT);
        circleNumber.createGraphics().drawImage(numbers.get((Integer.parseInt(array[0]))), x: 0, y: 0, observer: null);
        circleNumber.createGraphics().drawImage(Assets.numberSpace, x: 5, y: 0, observer: null);
        circleNumber.createGraphics().drawImage(numbers.get((Integer.parseInt(array[1]))), x: 6, y: 0, observer: null);
    }
}
```

The player number is slightly more complicated as it can be either single or double digit. Firstly the method checks if it's a single digit or double. If it's a single digit the single number is stored and the offset variable is set to 7. If it's a double digit number. It creates a new bufferedimage and adds both number images to this image along with a spacer which is a blank column from the number spreadsheet. These three images are combined into a single double digit number image. The offset value is then set to 5. When the number image is drawn to the screen this offset number tells the system where to draw the number image so that it is correctly centered on the circle image.

The main window of the program consists of a single JFrame with a canvas. All graphic images are drawn onto this canvas. An image of an ice rink is the first image drawn followed by the players and the puck. Rendering of these items are handled by the current game state, the implementation of which will be discussed later.

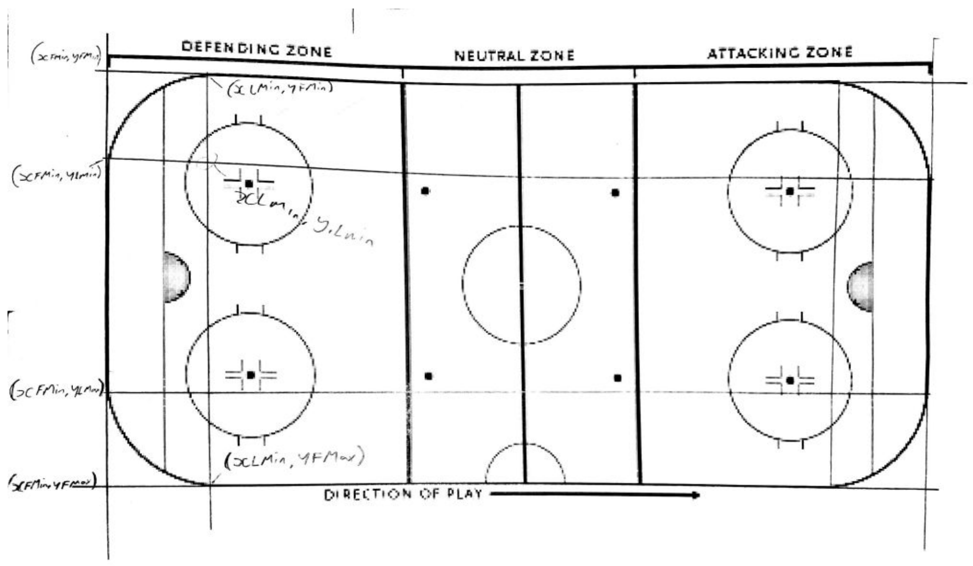
4.2.3. Position Calculations

The Arena class handles many aspects involving positions on the ice rink. The rink is split into 15 separate zones in each half of the rink. These zones are used to influence the players behaviour. Players in certain zones are more likely to shoot while in other zones they’re more likely to pass. Players have prefered zones that they will move to based on the position they play. A diagram of the zones can be seen below.

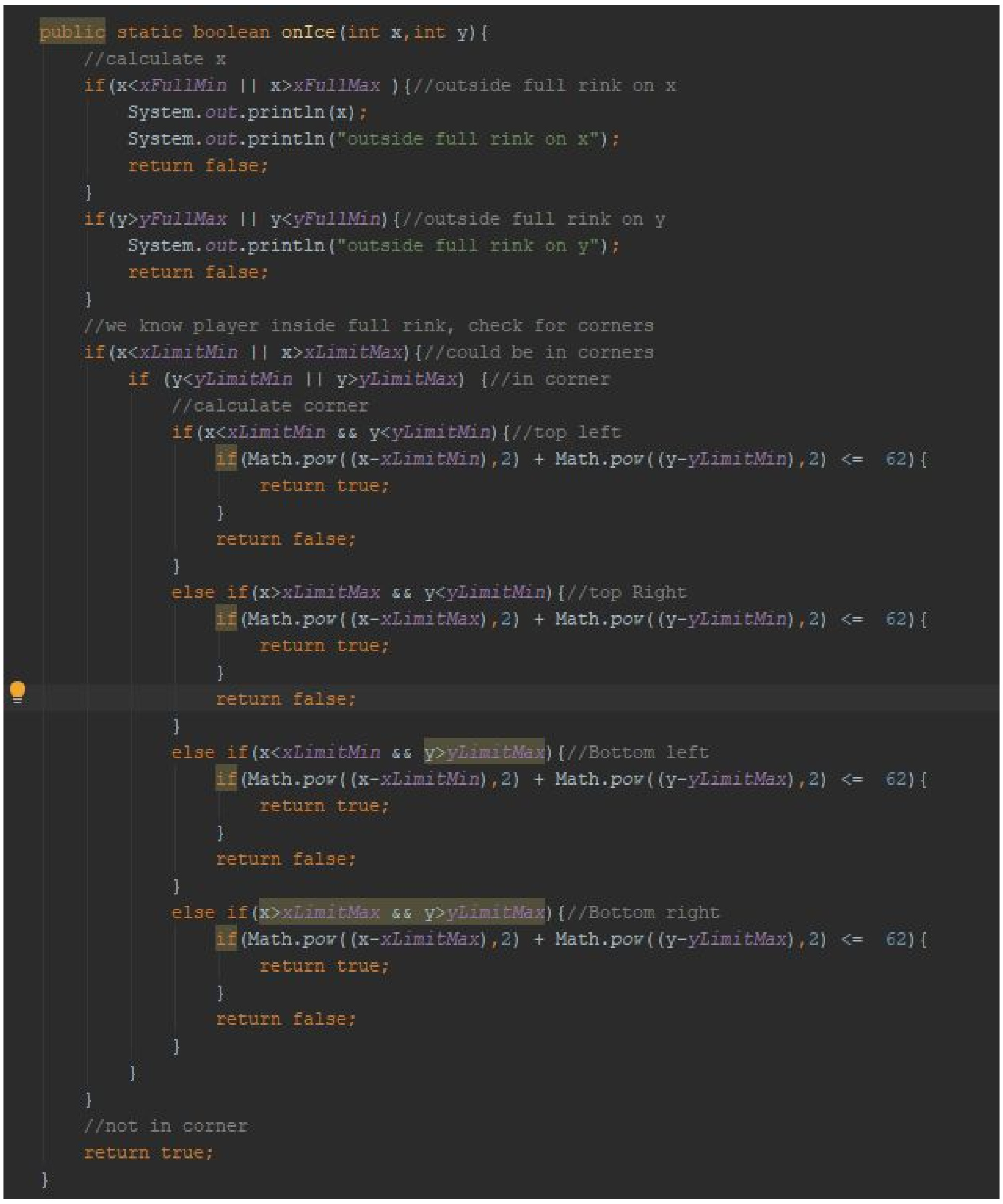


The position of each zone is stored as a distance from the centerline. Zones can be mirrored for offensive and defensive players.

The second function of the arena class is to ensure that all moves are legal and inside the rink. This is slightly more complex than other sports as the arena has rounded corners and the puck can not go out of play (apart from over the boards which is not implemented). To simplify this issue there is two lines on each axis, a full and a minimum line. These lines are used to separate the corners from the rest of the rink. The image below shows how these lines are used.



The code implementation for these checks first checks that the given coordinates are within the full rink bounds, this includes a small area that is outside the corners. If it is within this area it then checks if it is in one of the corners. It then uses the graph function of a circle that is approximately the same radius as the corner of the rink, if we are inside this circle we know we are still on the rink. If the coordinates is not in one of these corners we know that it is within the rink.



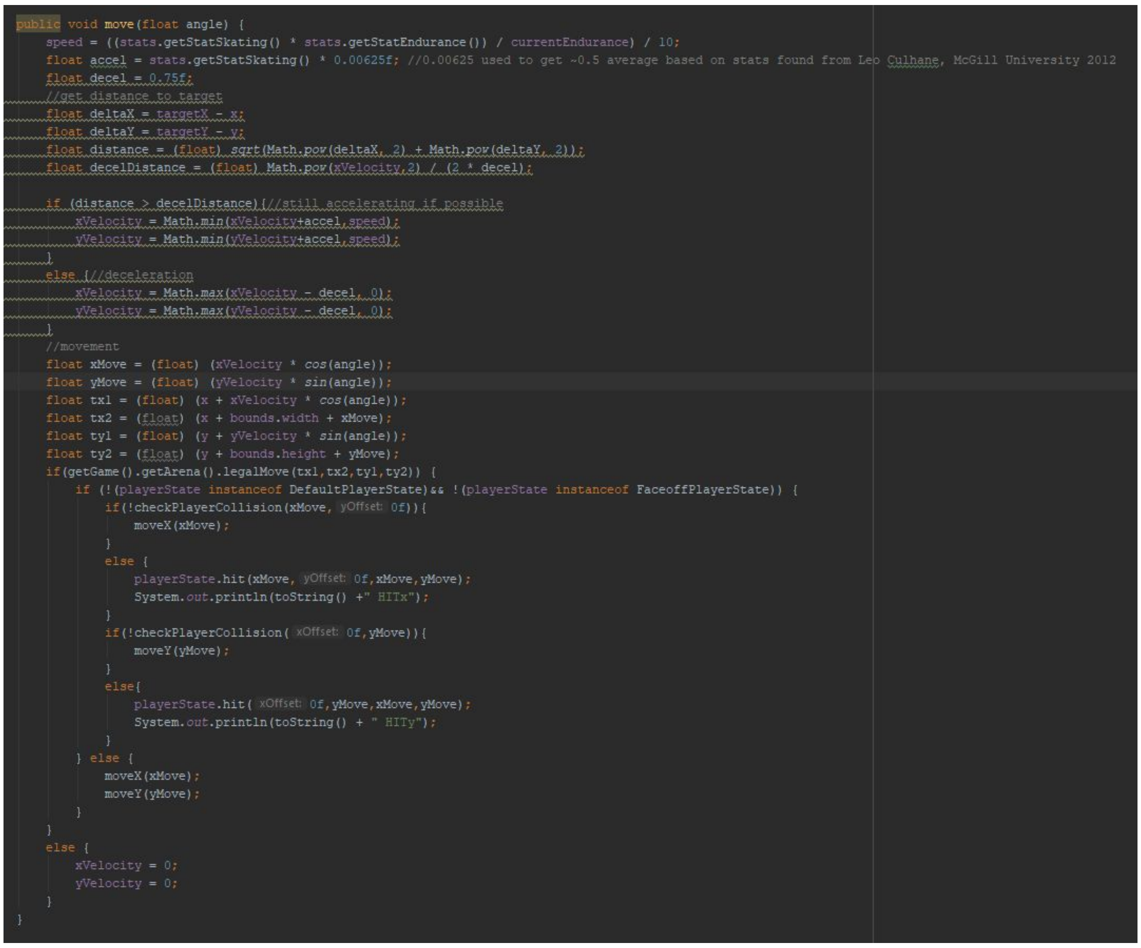
```java
public static boolean onIce(int x,int y){
    //calculate x
    if(x<xFullMin || x>xFullMax ){//outside full rink on x
        System.out.println(x);
        System.out.println("outside full rink on x");
        return false;
    }
    if(y>yFullMax || y<yFullMin){//outside full rink on y
        System.out.println("outside full rink on y");
        return false;
    }
    //we know player inside full rink, check for corners
    if(x<xLimitMin || x>xLimitMax){//could be in corners
        if (y<yLimitMin || y>yLimitMax) {//in corner
            //calculate corner
            if(x<xLimitMin && y<yLimitMin){//top left
                if(Math.pow((x-xLimitMin),2) + Math.pow((y-yLimitMin),2) <=  62){
                    return true;
                }
                return false;
            }
            else if(x>xLimitMax && y<yLimitMin){//top Right
                if(Math.pow((x-xLimitMax),2) + Math.pow((y-yLimitMin),2) <=  62){
                    return true;
                }
                return false;
            }
            else if(x<xLimitMin && y>yLimitMax){//Bottom left
                if(Math.pow((x-xLimitMin),2) + Math.pow((y-yLimitMax),2) <=  62){
                    return true;
                }
                return false;
            }
            else if(x>xLimitMax && y>yLimitMax){//Bottom right
                if(Math.pow((x-xLimitMax),2) + Math.pow((y-yLimitMax),2) <=  62){
                    return true;
                }
                return false;
            }
        }
    }
    //not in corner
    return true;
}
```

4.2.4. Entity Movement

All moving objects inherit from the entity class which contains the majority of the code for movement. However, the skater class contains some additional movement code which implements an acceleration and deceleration system. The speed of the puck is determined by the player passing, shooting, strength, and puck handling attributes. The speed of the player is determined by their speed and skating attributes.

The main feature of the movement system is the target position. Both players and the puck will have a target position they will move towards. However in the case of the puck only the angle to this position is used for the movement. This ensures that the puck keeps moving after it starts. The current puck movement system is not perfect in the way the puck bounces off the boards of the arena but for the majority of situations it works fine.

Player movement involves a basic acceleration and deceleration system. Unlike the puck the player movement checks for the direction to the target at every move. The distance to the target is calculated and the distance required to decelerate is calculated. If the player still have time to accelerate to their maximum speed they do, otherwise they start to slow down to stop at their target position. The code for this can be seen below.

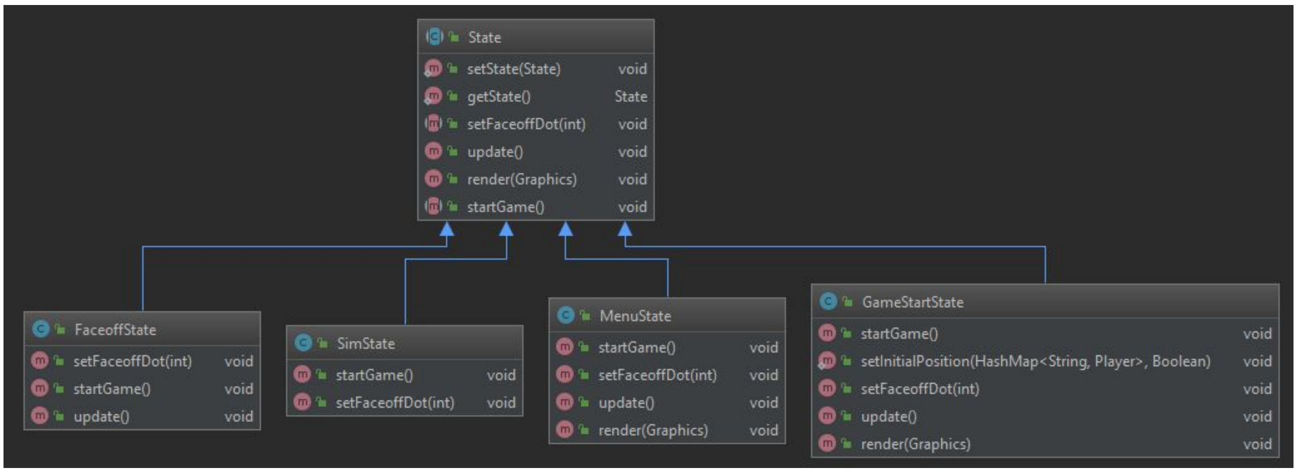


```java
public void move(float angle) {
    speed = ((stats.getStatSkating() * stats.getStatEndurance()) / currentEndurance) / 10;
    float accel = stats.getStatSkating() * 0.00625f; //0.00625 used to get ~0.5 average based on stats found from Leo Culhane, McGill University 2012
    float decel = 0.75f;
    //get distance to target
    float deltaX = targetX - x;
    float deltaY = targetY - y;
    float distance = (float) sqrt(Math.pow(deltaX, 2) + Math.pow(deltaY, 2));
    float decelDistance = (float) Math.pow(xVelocity,2) / (2 * decel);

    if (distance > decelDistance){//still accelerating if possible
        xVelocity = Math.min(xVelocity+accel,speed);
        yVelocity = Math.min(yVelocity+accel,speed);
    }
    else {//deceleration
        xVelocity = Math.max(xVelocity - decel, 0);
        yVelocity = Math.max(yVelocity - decel, 0);
    }
    //movement
    float xMove = (float) (xVelocity * cos(angle));
    float yMove = (float) (yVelocity * sin(angle));
    float tx1 = (float) (x + xVelocity * cos(angle));
    float tx2 = (float) (x + bounds.width + xMove);
    float ty1 = (float) (y + yVelocity * sin(angle));
    float ty2 = (float) (y + bounds.height + yMove);
    if(getGame().getArena().legalMove(tx1,tx2,ty1,ty2)) {
        if (!(playerState instanceof DefaultPlayerState)&& !(playerState instanceof FaceoffPlayerState)) {
            if(!checkPlayerCollision(xMove, yOffset: 0f)){
                moveX(xMove);
            }
            else {
                playerState.hit(xMove, yOffset: 0f,xMove,yMove);
                System.out.println(toString() +" HITx");
            }
            if(!checkPlayerCollision( xOffset: 0f,yMove)){
                moveY(yMove);
            }
            else{
                playerState.hit( xOffset: 0f,yMove,xMove,yMove);
                System.out.println(toString() + " HITy");
            }
        } else {
            moveX(xMove);
            moveY(yMove);
        }
    }
    else {
        xVelocity = 0;
        yVelocity = 0;
    }
}
```

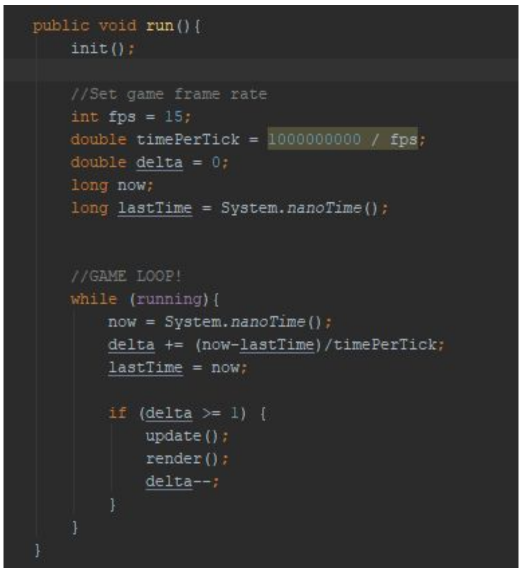
When setting a players target position there is two possible methods that can be used. “setTargetPositionAbsolute” and “setTargetPositionRelative”. The setTargetPositionAbsolute sets the target position to the x,y coordinate given to it, whereas setTargetPositionRelative uses a distance from the centerline of the ice rink and the home or away end of the rink.

4.2.5. Game States and ticks

The main running of the actual simulation is handled by a number of game states. These game states use a mix of the state design pattern and the template design pattern. The main Sim class contains a single GameState which will be one of “MenuState”, “GameStartState”, “FaceoffState “, and “SimState” which can be seen in the UML diagram below. These states all contain the “update” and “render” methods. Both of these methods run on every tick of the simulation. The update method is used to run every players behaviour and the pucks every tick. The render method then redraws the players and puck to the screen with their current position.



The simulation runs on a tick based system. Every tick each player will take in data about their surroundings and current game information and then perform an action based on this. The default tick rate is 15 ticks per second. I found this to be a good tick rate to be able to see the action take place at a realistic speed. It would be possible to either increase or decrease the tick rate but changing the TPS variable in the code. This would allow the user to speed through a simulation.



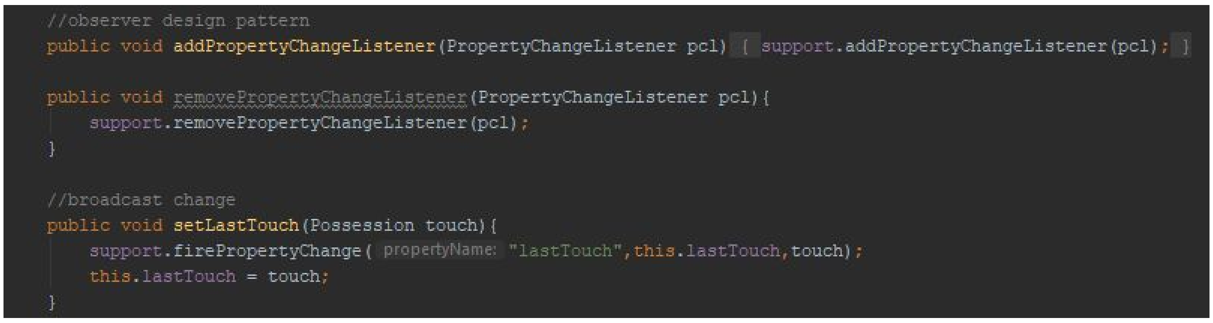
```
public void run(){
    init();

    //Set game frame rate
    int fps = 15;
    double timePerTick = 1000000000 / fps;
    double delta = 0;
    long now;
    long lastTime = System.nanoTime();


    //GAME LOOP!
    while (running){
        now = System.nanoTime();
        delta += (now-lastTime)/timePerTick;
        lastTime = now;

        if (delta >= 1) {
            update();
            render();
            delta--;
        }
    }
}
```

4.2.6. Puck Handling

The game puck has four possible states that it could be at any point in the game: It could be in the faceoff state where neither team are actively trying to get the puck until the faceoff is completed; The puck could be loose where no player has possession and both teams are attempting to get the puck; Or the puck could be possessed by a player on either the home team or the away team. The state of the puck will decide how players act in the simulation therefore it is vital that all players have the correct puck state at all points during the simulation.

To have players away of the puck's current state I have implemented the observer design pattern. The observer pattern is a behavioral design pattern that is used when there is one "subject" class (the puck) and multiple observer classes (the players). When there is a change to the subject this is broadcast to any number of observers.

The puck has a method "setLastTouch" where one of the possible puck states is passed as an enum type. When this method is executed it broadcasts this change to all players and they can change their behavior based on this new puck state. The implementation of player behaviour will be discussed in the next section.



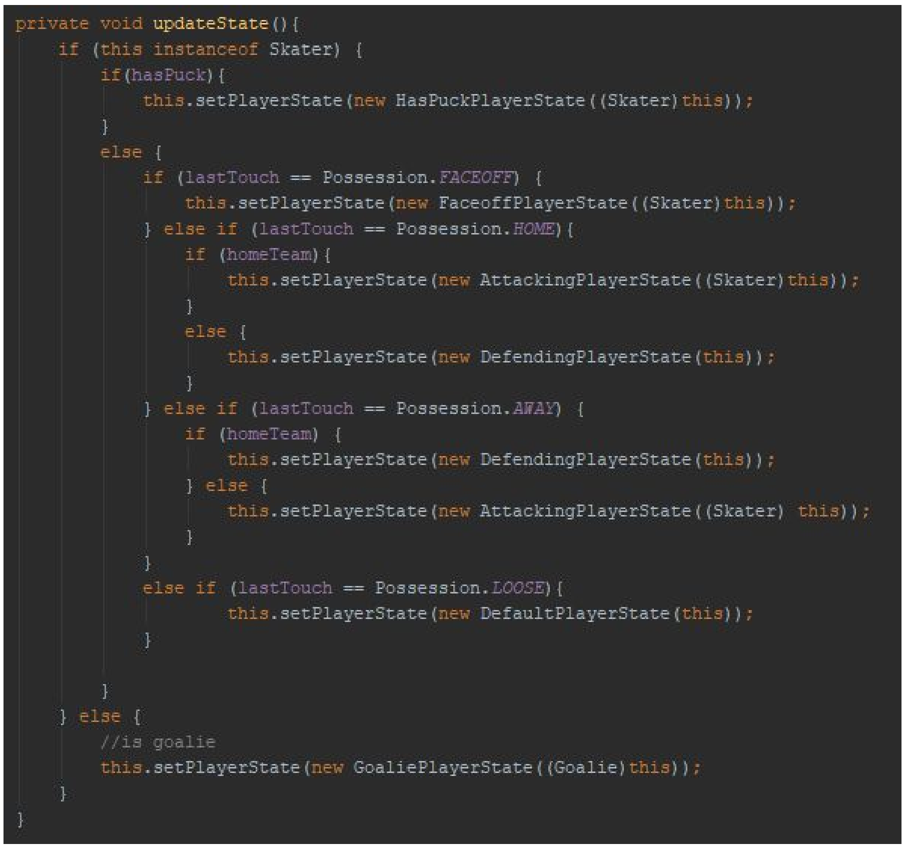
```java
//observer design pattern
public void addPropertyChangeListener(PropertyChangeListener pcl) { support.addPropertyChangeListener(pcl); }

public void removePropertyChangeListener(PropertyChangeListener pcl){
    support.removePropertyChangeListener(pcl);
}

//broadcast change
public void setLastTouch(Possession touch){
    support.firePropertyChange( propertyName: "lastTouch",this.lastTouch,touch);
    this.lastTouch = touch;
}
```

The implementation of this design pattern uses the java property change API. The puck class contains a PropertyChangeSupport object. This handles the broadcasting of the puck update. Also within the puck class is two methods "addPropertyChangeListener" and "removePropertyChangeListener", these manage the adding and removing of the players that listen or observer the change.

The player classes implement the PropertyChangeListener API and when a player object is created runs the "addPropertyChangeListener" method to add itself as an observer. The player class also contains a method called "propertyChange", this is run when a puck state change is broadcast. When executed it gets the new puck state and updates the local variable with the new value and forces an update to the player state which will be discussed next.

4.2.7. Player States

Players have states similar to how game states have been implemented. These player states define a players behavior at any one moment. The current player state is decided by the current state of the puck. The possible player states are "defaultPlayerState", "faceoffPlayerState", "attactingPlayerState", "defendingPlayerState", "hasPuckPlayerState", "goaliePlayerState".



```java
private void updateState(){
    if (this instanceof Skater) {
        if(hasPuck){
            this.setPlayerState(new HasPuckPlayerState((Skater)this));
        }
        else {
            if (lastTouch == Possession.FACEOFF) {
                this.setPlayerState(new FaceoffPlayerState((Skater)this));
            } else if (lastTouch == Possession.HOME){
                if (homeTeam){
                    this.setPlayerState(new AttackingPlayerState((Skater)this));
                }
                else {
                    this.setPlayerState(new DefendingPlayerState(this));
                }
            } else if (lastTouch == Possession.AWAY) {
                if (homeTeam) {
                    this.setPlayerState(new DefendingPlayerState(this));
                } else {
                    this.setPlayerState(new AttackingPlayerState((Skater) this));
                }
            }
            else if (lastTouch == Possession.LOOSE){
                    this.setPlayerState(new DefaultPlayerState(this));
            }

        }
    } else {
        //is goalie
        this.setPlayerState(new GoaliePlayerState((Goalie)this));
    }
}
```

Every game tick the updateState method is run to ensure that the player has the correct behaviour for the current conditions. Each playerstate has two main methods, “think()” and “act”. The think method is where players gather all the data about their surrounding and make decisions on what to do. These decisions are driven by the gathered data, player attributes, and a random element. Once a decision has been made the act method performs the action along with any other actions needed for that behaviour.

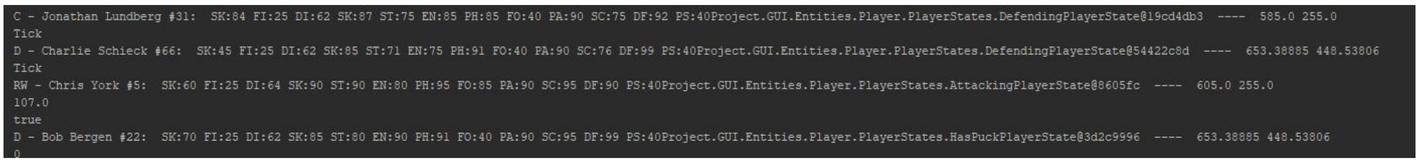
5. Testing

5.1. Challenges of Testing a GUI

Testing this system has been very challenging, the combination of the GUI and the number of random elements means that most traditional testing methods would not be appropriate for this project. Testing on this system has largely been reliant on breaking the system into small components and finding ways to test them separately before combining with other components. The best example of this would be player behaviour. Each player state was tested with no interactions from other players or just the players necessary to test that specific behaviour.

5.2. System Testing and Scenarios

When testing the system it was important to know the current state of every player. I made extensive toString methods for all player and puck classes which included all the information to be able to view how decisions were being made. I created a function “Random” in the game class so I could easily turn off or on the random aspect to all decision making processes to allow further control of the simulation for testing. While this was not a perfect solution it did provide a way to test the basic functionality of the software. In addition to this the debugging tool from intelliJ proved invaluable for debugging the code. Despite this testing a number of bugs and issues still remain which I am unable to fully track down. Reproducing issues is difficult given the nature of the program even with these modifications.



```
C - Jonathan Lundberg #31:  SK:84 FI:25 DI:62 SK:87 ST:75 EN:85 PH:85 FO:40 PA:90 SC:75 DF:92 PS:40Project.GUI.Entities.Player.PlayerStates.DefendingPlayerState@19cd4db3  ----  585.0 255.0
Tick
D - Charlie Schieck #66:  SK:45 FI:25 DI:62 SK:85 ST:71 EN:75 PH:91 FO:40 PA:90 SC:76 DF:99 PS:40Project.GUI.Entities.Player.PlayerStates.DefendingPlayerState@54422c8d  ----  653.38885 448.53806
Tick
RW - Chris York #5:  SK:60 FI:25 DI:64 SK:90 ST:90 EN:80 PH:95 FO:85 PA:90 SC:95 DF:90 PS:40Project.GUI.Entities.Player.PlayerStates.AttackingPlayerState@8605fc  ----  605.0 255.0
107.0
true
D - Bob Bergen #22:  SK:70 FI:25 DI:62 SK:85 ST:80 EN:90 PH:91 FO:40 PA:90 SC:95 DF:99 PS:40Project.GUI.Entities.Player.PlayerStates.HasPuckPlayerState@3d2c9996  ----  653.38885 448.53806
0
```

6. Evaluation

6.1. System Successes

The final system does work as a hockey simulation engine, a number of my initial aims and objectives have been met. I believe the biggest success however was the amount I have learnt throughout the process. However this is also a reason that a number of my aims and objectives were not met. The aspect of the system I am most happy with is the graphical display, I had a number of worries about this going into the project but I feel the finished product works very well and achieves everything that I wanted it to.

I'm happy with how the player system worked out, the use of the player states to implement player behaviour allows for a lot of flexibility in the simulation. Some slight improvements can be made to how the state system was implemented but I feel it was the correct approach for this problem. Despite the challenges I faced during the project I am happy with the result.

6.2. System Failures

The final system did fall short of a number of the initial aims, during the process of creating the system I had to scrap certain features that I initially had planned. My original goals overestimated the amount of work that would have to go into the system for even the most basic behaviours. The biggest issue with the finished system is the movement system and more specifically players tracking and intercepting the puck. I attempted multiple methods to implement this yet could not get a system that worked as I would like. A large part of the simulation relies on this behaviour so this not working as hoped has a large impact on the whole system.

A lack of more detailed planning caused a number of issues throughout the project. There was numerous occasion when implementing features that I had to go back to parts of the codebase I thought were complete and either heavily edit or even rewrite them so that future features would be able to be implemented. This took up a lot of time which could have been spent improving the simulation behaviour.

Overall time management is also an area that could have been improved. Towards the beginning of the project is where my time management struggled. I had a tendency to spend too much time perfecting small areas of the system when I had much more important features that needed attention. As the project progressed I did begin to address this by setting myself shorter and more detailed deadlines. I feel if I had this in place from the beginning the overall system would be better than the current system.

Due to the issues previously mentioned and also underestimating the time required for the system overall there are a number of features missing that I would have liked to implement. These features include having a full team of 21 players with substitutions, a full detailed hitting/collision system, a penalty system, and a better puck/goalie interaction system.

6.3. Future Work

If I had more time to work on this project my first priority would be to finish the implementation of all the features I initially had planned. The next addition to the program I would make is to add the ability to simulate multiple games and record statistics over multiple games. However, the majority of this future work would be rather difficult to implement into the current codebase.

6.4. A Second Attempt

This project has taught me a lot about the development of a larger java system. I’ve learnt the importance of having a detailed design and plan before starting the implementation. I found that lack of planning led to a number of issues where it was either impossible or much harder to implement a new feature due to how other areas of the code was implemented.

I would also reduce the number of classes the system has, the current system contains a number of classes that are almost redundant or fulfil functions that could very easily be handled by other existing classes

6.5. Conclusion

In conclusion the system fulfills the simple aim of providing an ice hockey simulation with a graphical representation. I overestimated what I would be able to produce within the time limits of this project and with my current skill levels which led to numerous features having to be cut from the finished system. The project has taught me a lot about myself and about developing large software projects with java. This is a system I will revisit in the future to improve on.

Appendices

Appendix A - What is Ice Hockey

Source - Ice Hockey UK

“Ice hockey is played in over 55 countries worldwide from Andorra to Yugoslavia. The world governing body, the International Ice Hockey Federation (IIHF), was organised in 1908, with Great Britain being a founder member.

In 1903, Great Britain had a five-team league and the first Scottish game was in 1908 in Crossmyloof, Glasgow.
The British Ice Hockey Association (BIHA) was formed in 1914 and was wound down in 1999, when it was taken over by Ice Hockey UK.

The first European Championship was won by Great Britain in 1910 and the first World Cup was won by Canada in 1920. Great Britain were Olympic champions in 1936.

Two linesmen and one or two referees – depending on the league – control the game. A game is divided into three 20-minute periods of actual playing time, divided by two intervals of usually around 15 minutes. When a whistle is blown the clock stops.

Six players from each team are on the ice at any one time. The line up being; netminder, two defencemen and three forwards. These players can be changed at any time as the game is played at such a speed.

A team is usually made up of between 17 and 22 players. This consists of two netminders, four to six in the defence and three to four lines of forwards.

The netminder will usually remain unchanged throughout the game, but it is quite common for him to leave the ice during the last seconds of a game.
This means there is no goalie and six outskaters on the ice in a risk-all effort to score. This usually happens at the end of a game where teams are separated by a single goal and can be very exciting.

Play begins with a face-off when the referee drops the puck in the centre circle between the sticks of the two centres. Other face-offs happen after a goal is scored and at other times after misplay. The puck only becomes dead when the whistle blows or it is hit over the barrier.

A goal is scored when the puck enters the net or goes across the goal line propelled by a stick. If the puck is kicked or thrown into the net there is no goal. Goal judges sit behind each goal and switch on a red light when a goal is scored. The goals are 1.22m (4′) tall and 1.83m (6′) wide.

The puck is circular, made of solid vulcanised rubber, is 7.62 cm (3″) in diameter, 2.54cm (1″) thick and weighs 143 grams (5½ ounces). A player may stop the puck with hand, body or skate at any time in any position. The puck must not be pushed forward except by skate or stick.

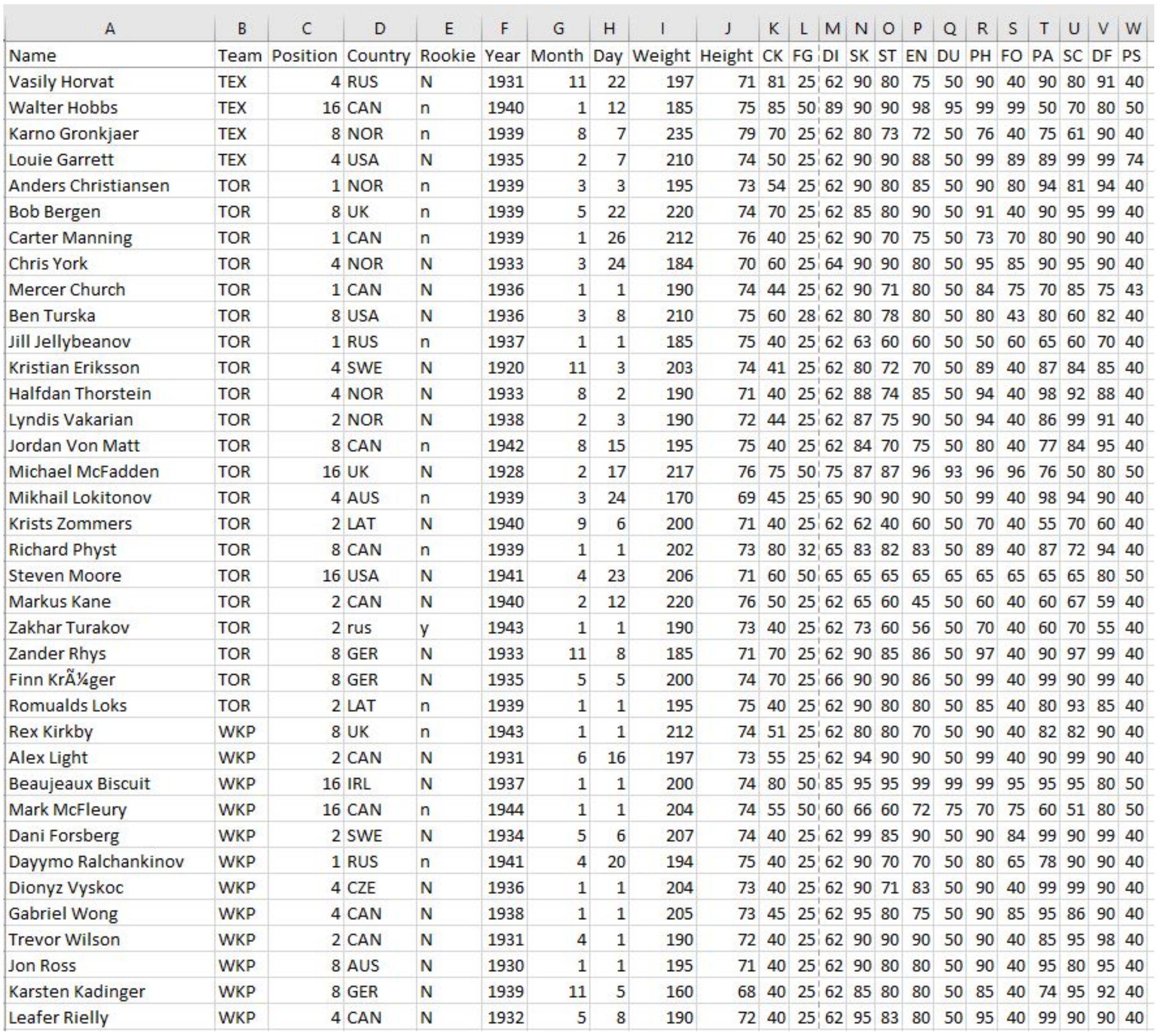
The space between goals is divided by blue lines into three zones; defence, neutral and attacking zones – each is a third of the playing area.
An attacking player may only enter the attacking zone behind the puck or puck-carrier. A pass cannot be made to an attacker from a team-mate who is outside the attacking zone. This is known as being offside and the whistle will be blown and a face off taken.

Players are penalised for infringements of rules by being sent off the ice for two or more minutes according to the severity of the offence and no substitute can be made for the duration of the penalty, which he serves in a special penalty box.
The goalie does not serve his own penalties but a team-mate must sit in the box for most penalties given against him, except if he is thrown out of the game.
Offences include charging, elbowing, boarding, tripping, checking from behind, high sticks, interference, roughing and unsportsmanlike conduct.

Ice hockey players wear specialised equipment, which is specifically designed for safety. Players are well padded – they wear knee pads, shin, hip, elbow and shoulder guards, thick gauntlet-type gloves, long stockings that fit over the knee pads, padded shorts that lace up at the front and sweaters in team colours over everything.

Boots are stronger and different from figure skating boots. They have lower ankle support, reinforced toes and padded tongues. The blade has a plain point, is straight and narrow and now hollow ground. The two upright stanchions are higher on a hockey skate changing the centre of gravity.
Ice hockey, the world's fastest team game, is full of skilful stick handling, tactics, speed and grit."

Appendix B - CSV File Data



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Name	Team	Position	Country	Rookie	Year	Month	Day	Weight	Height	CK	FG	DI	SK	ST	EN	DU	PH	FO	PA	SC	DF	PS
Vasily Horvat	TEX	4	RUS	N	1931	11	22	197	71	81	25	62	90	80	75	50	90	40	90	80	91	40
Walter Hobbs	TEX	16	CAN	n	1940	1	12	185	75	85	50	89	90	90	98	95	99	99	50	70	80	50
Karno Gronkjaer	TEX	8	NOR	n	1939	8	7	235	79	70	25	62	80	73	72	50	76	40	75	61	90	40
Louie Garrett	TEX	4	USA	N	1935	2	7	210	74	50	25	62	90	90	88	50	99	89	89	99	99	74
Anders Christiansen	TOR	1	NOR	n	1939	3	3	195	73	54	25	62	90	80	85	50	90	80	94	81	94	40
Bob Bergen	TOR	8	UK	n	1939	5	22	220	74	70	25	62	85	80	90	50	91	40	90	95	99	40
Carter Manning	TOR	1	CAN	n	1939	1	26	212	76	40	25	62	90	70	75	50	73	70	80	90	90	40
Chris York	TOR	4	NOR	N	1933	3	24	184	70	60	25	64	90	90	80	50	95	85	90	95	90	40
Mercer Church	TOR	1	CAN	N	1936	1	1	190	74	44	25	62	90	71	80	50	84	75	70	85	75	43
Ben Turska	TOR	8	USA	N	1936	3	8	210	75	60	28	62	80	78	80	50	80	43	80	60	82	40
Jill Jellybeanov	TOR	1	RUS	n	1937	1	1	185	75	40	25	62	63	60	60	50	50	60	65	60	70	40
Kristian Eriksson	TOR	4	SWE	N	1920	11	3	203	74	41	25	62	80	72	70	50	89	40	87	84	85	40
Halfdan Thorstein	TOR	4	NOR	N	1933	8	2	190	71	40	25	62	88	74	85	50	94	40	98	92	88	40
Lyndis Vakarian	TOR	2	NOR	N	1938	2	3	190	72	44	25	62	87	75	90	50	94	40	86	99	91	40
Jordan Von Matt	TOR	8	CAN	n	1942	8	15	195	75	40	25	62	84	70	75	50	80	40	77	84	95	40
Michael McFadden	TOR	16	UK	N	1928	2	17	217	76	75	50	75	87	87	96	93	96	96	76	50	80	50
Mikhail Lokitonov	TOR	4	AUS	n	1939	3	24	170	69	45	25	65	90	90	90	50	99	40	98	94	90	40
Krists Zommers	TOR	2	LAT	N	1940	9	6	200	71	40	25	62	62	40	60	50	70	40	55	70	60	40
Richard Physt	TOR	8	CAN	n	1939	1	1	202	73	80	32	65	83	82	83	50	89	40	87	72	94	40
Steven Moore	TOR	16	USA	N	1941	4	23	206	71	60	50	65	65	65	65	65	65	65	65	65	80	50
Markus Kane	TOR	2	CAN	N	1940	2	12	220	76	50	25	62	65	60	45	50	60	40	60	67	59	40
Zakhar Turakov	TOR	2	rus	y	1943	1	1	190	73	40	25	62	73	60	56	50	70	40	60	70	55	40
Zander Rhys	TOR	8	GER	N	1933	11	8	185	71	70	25	62	90	85	86	50	97	40	90	97	99	40
Finn KrÃ¼ger	TOR	8	GER	N	1935	5	5	200	74	70	25	66	90	90	86	50	99	40	99	90	99	40
Romualds Loks	TOR	2	LAT	n	1939	1	1	195	75	40	25	62	90	80	80	50	85	40	80	93	85	40
Rex Kirkby	WKP	8	UK	n	1943	1	1	212	74	51	25	62	80	80	70	50	90	40	82	82	90	40
Alex Light	WKP	2	CAN	N	1931	6	16	197	73	55	25	62	94	90	90	50	99	40	90	99	90	40
Beaujeaux Biscuit	WKP	16	IRL	N	1937	1	1	200	74	80	50	85	95	95	99	99	99	95	95	95	80	50
Mark McFleury	WKP	16	CAN	n	1944	1	1	204	74	55	50	60	66	60	72	75	70	75	60	51	80	50
Dani Forsberg	WKP	2	SWE	N	1934	5	6	207	74	40	25	62	99	85	90	50	90	84	99	90	99	40
Dayymo Ralchankinov	WKP	1	RUS	n	1941	4	20	194	75	40	25	62	90	70	70	50	80	65	78	90	90	40
Dionyz Vyskoc	WKP	4	CZE	N	1936	1	1	204	73	40	25	62	90	71	83	50	90	40	99	99	90	40
Gabriel Wong	WKP	4	CAN	N	1938	1	1	205	73	45	25	62	95	80	75	50	90	85	95	86	90	40
Trevor Wilson	WKP	2	CAN	N	1931	4	1	190	72	40	25	62	90	90	90	50	90	40	85	95	98	40
Jon Ross	WKP	8	AUS	N	1930	1	1	195	71	40	25	62	90	80	80	50	90	40	95	80	95	40
Karsten Kadinger	WKP	8	GER	N	1939	11	5	160	68	40	25	62	85	80	80	50	85	40	74	95	92	40
Leafer Rielly	WKP	4	CAN	N	1932	5	8	190	72	40	25	62	95	83	80	50	95	40	99	90	90	40

Appendix C - Attribute Descriptions

Source - ORIGINSZ (2013)

“Skater Ratings:

CK= Checking
Effects the Usefulness: Endurance, Skating
Starts At: 40

Checking rating is not how hard your player hits, it is in fact how often your player goes for hit. Checking accuracy and frequency is also affected by 2 other ratings as stated above. Endurance is exactly what most think it is how long and hard your player can play. So having a low endurance but high checking causes your player to be less effective in other areas as he wears down easier throwing the body. Skating affects speed ability to turn and your ability to catch people because if you can't catch them, you are less likely to hit someone faster than your player.

FG= Fighting
Effects the Usefulness: Endurance, Strength
Starts At: 25

Fighting is not, like many believe, your ability to win fights but more on the frequency on which your guy drops the gloves. Endurance and Strength only play a minor role both in which help only slightly your ability to win or draw a fight in which you may not be able to win.

DI= Discipline
Effects the Usefulness: Endurance, Skating, Checking
Starts At: 62

Discipline is the ability to take less penalties. The higher the discipline, the better your chance is to avoid taking penalties. Endurance and Skating, like checking, effect this skill as well. If your player is playing big minutes but is always tired he's more prone to taking a dumb penalty. Skating comes into play with holding and tripping etc. Generally if someone has a much higher skating rating and your guy gets blown past, he may be more prone to taking bad PIMs trying to stop him.

SK= Skating
Effects the usefulness: Endurance, Strength
Starts At: 40

Skating is the ability to turn, skate faster and keep yourself in a play. Once again, a major factor is endurance as if your player is tired, even if he has 80 skating and the other guy has 60, he will get left behind if that player has higher endurance. Strength also plays a minor factor into skating if your stronger you can get up to speed faster than someone with lower strength.

Added: Part of the STHS decision making process (more on this following all skater attributes), this stat helps dictate how often your player will carry the puck. This kicks in and factors in odds to make the pass, shoot, or keep moving. If STHS feels your best odds are to keep skating a higher skating will allow you to keep control of the puck.

ST= Strength
Effects the usefulness: Endurance, Skating, Faceoffs
Starts At: 40

Strength is your ability to stay on your feet with a check, to out muscle someone else for the puck, and your ability to knock your opponent off the puck. Again endurance plays a factor in this skill - if you're tired even with a high strength you can be outmatched by someone with lower strength who may not be tired. Skating is a minor factor the better skater you are the less likely with high strength you are to be knocked off the puck.

EN= Endurance
Effects the usefulness: Nothing
Starts At: 40

Endurance is a key stat that affects basically everything. It does what you would expect: it allows you to play big/tougher minutes. This is a stat everyone should focus on.

DU= Durability*- In the SMJHL/SHL we do not get injured
Effects the usefulness: Endurance, Strength.
Starts At: 40

Injuries are currently turned off as of S25.

PH= Puck Handling
Effects the usefulness: Endurance, Strength, Skating
Starts At: 40

Puck Handling is your ability to create plays, but its main focus in the sim is the ability to prevent the puck from being stolen off of you. The higher your puck handling the more likely you will keep control of the puck and can strip it from someone else. This is effected by 3 skills: Endurance (which is plain and simple, if you're tired its easier to get the puck off of you), Strength (again pretty straight forward if you're stronger on your stick its harder to strip the puck from your player), and last but not least, Skating (if your guy is slower than his opponent it gives the player more time to strip the puck off of you).

Added: Has a small effect on your ability to tip the puck when standing in front(almost like hand eye). Part of the STHS decision making process, This stat helps dictate how often your player will carry the puck. This kicks in and factors in odds to make the pass, shoot, or keep moving. If STHS feels your best odds are to keep skating a higher skating will allow you to keep control of the puck.

FO=Faceoffs
Effects the usefulness: Endurance, Strength
Starts At: 40

The ability to win face offs. Plain and simple the higher it is the more likely you are to win. Endurance is like everything, if your player is tired taking a draw he is more likely to lose it to a fresh or less tired center of similar face off rating. Strength: if you're not tired and have similar face off skill this helps a bit allowing you to perhaps out muscle your opponent.

PA= Passing
Effects the usefulness: Line mates, Your SC rating
Starts At: 40

Passing is not really your ability to make great passes but more the frequency is which your player will look to make the pass. This rating is effected not by you but a lot of the time by line mates. If you have a line mate with a higher scoring rating the passer will attempt more passes to that player. The other factor that comes in is if another player on similar line has a higher SC but similar passing than the other he will actually attempt more shots on goal than he would with a pure scoring winger.

Added: Part of the decision making process, just a bit more in depth in this one. Every encounter in STHS bases on skate with the puck, pass, or shoot. A higher passing means you are more likely to make the pass at the right time, versus passing into your opponents stick.

SC= Scoring
Effects the usefulness: Line mates, Your PA rating
Starts At: 40

Scoring is not your ability to score but, like passing, the frequency of which you shoot the puck. Like passing, if your line mate is a better passer and similar scorer your guy will shoot more than his even though you're both scorers. If you have a higher passing rating than line mate you will actually attempt passes more than usual.

Added: Part of the decision making process, just a bit more in depth in this one. Every encounter in STHS, bases on skate with the puck, pass, or shoot. A higher passing means you are more likely to make the shoot at the right time, versus shoot and the Goalie make a save, or an easy turn over.

DF=Defense
Effects the usefulness: Endurance, Strength
Starts At: 40

This effects your in zone positioning and your frequency to block shot and make the right defensive play. This also effects the ability to strip the puck in the defensive zone. Strength plays a minor role in allowing you the ability to move people from the front of your net or to knock someone off the puck. Endurance, if you're tired you are less likely to get back in position on defense.

Added: Part of the Decision making process, again just more in depth here it effects your players ability when defending to make the right choice. If that's block the shot, make the hit, poke check, or just keeping him out of the zone. DF augments the Passing part of the decision making process.

PS= Penalty shot
Effects the usefulness: Goalie PS rating
Starts At: 40

Your ability to score on shoot out or penalty shot goals this is also effected by a goalies similar stat and luck of the draw.

The Decision Making Process
When your player has the puck they have three options; Pass, Shoot, or Carry. Each of these options are represented by PA, SC, SK. Their ratings will determine how often you attempt something as well as determine how well you do it. PA:50, SC:50, SK:50 is just as likely as PA:99, SC:99, SK:99 to choose any task. However, PA:99, SC:99, SK:99 will perform those tasks better. Keep this in mind when you're building your player. A forward line who are only spec'd with high SC is less likely to perform well against a line that has balance, or a line who has players who fit specific roles. A line of three Wayne Gretzky's will get run over by a line of McSorley, Kurri, and Gretzky.

Goalie Ratings

SK= Skating
Effects the usefulness: Endurance, Agility

Skating is your ability to get across for goals and to retrieve a dump in to start a play. This is effected by endurance if your tired you will move slower than a higher endurance rating. The higher your agility effects this as well as it effects your ability to turn and go side to side. Secondary skill

DU= Durability*- In the SMJHL/SHL we do not get injured.
Effects the usefulness: Endurance

The higher it is the less you get injured plain and simple. If you are tired you are more likely to be injured as well so endurance is a factor.

EN= Endurance
Effects the usefulness: Nothing

Your ability to stop pucks plain and simple also your ability to play back to back games at a higher level than a goalie with low endurance doing the same. The higher your endurance the more consistent your player will be.

SZ=Size
Effects the usefulness: Endurance

Size is another factor in your ability to stop a puck. Size is not how big your player is but his ability to look bigger in the net to make the holes behind him appear smaller to the shooter. If your tired your guy may challenge less opening up the net more. Secondary skill

AG= Agility
Effects the usefulness: Endurance, Skating

Your ability to move side to side and turn on a dime which helps stop more shots. Again if your player is tired he moves slower side to side and even turning. Skating also effects this is the ability to turn and set yourself up for the shot. Secondary skill

RB= Rebound Control
Effects the usefulness: Endurance

Its what you would expect the ability to control rebounds and make them less dangerous for your team. If your player is tired he is more prone to bad rebounds than he would fresh.

SC= Style Control
Effects the usefulness: Endurance

Your ability to keep calm and cool is a primary skill in the ability to stop pucks. Not a whole lot else again if your tired it will effect your ability to stay focused later in the game.

HS= Hand Speed
Effects the usefulness: Endurance, Reaction Time

Your ability to get a glove or blocker up quickly it also effects covering the puck up more frequently. As usual if your tired you will move slower and not be able to get your glove up quick enough. Reaction time also plays a role in this if your player is slow to react hes more prone to getting beat. Primary stat

RT= Reaction Time
Effects the usefulness: Endurance, Hand Speed

Your ability to read and react to plays this is one of the best skills a goalie can have if he wishes to be a top goalie it effects almost everything. Being tired and slow hands can effect the usefulness of this skill.

PH= Puck Handling
Effects the usefulness: Endurance, Skating

The frequency in which your player will play or go to retrieve a puck behind the net aswell as the ability to corral it rather than fumble it and turn it over. Endurance and skating effect this is he cant skate he may get to the puck to late same if he is tired.

PS= Penalty Shot
Effects the usefulness: Skaters Penalty shot

Your ability to stop penalty shots and shoot outs this is effected by luck and the opponents similar rating.”

References

Ice Hockey UK 2019, About Ice Hockey, Accessed 10/04/2019 <https://www.icehockeyuk.co.uk/about-ice-hockey/>

OriginSZ 2013, Attribute Descriptions How Each Attribute Affects Your Player, Accessed 20/04/2019 <http://simulationhockey.com/showthread.php?tid=22263>

Bibliography

www.tutorialspoint.com. (2019). *Design Patterns Observer Pattern*. [online] Available at: https://www.tutorialspoint.com/design_pattern/observer_pattern.htm [Accessed 8 Feb. 2019].

www.tutorialspoint.com. (2019). *Design Patterns Singleton Pattern*. [online] Available at: https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm [Accessed 8 Feb. 2019].

www.tutorialspoint.com. (2019). *Design Patterns State Pattern*. [online] Available at: https://www.tutorialspoint.com/design_pattern/state_pattern.htm [Accessed 8 Feb. 2019].