NETWORK PROBE SELECTION PROBLEM

PROJECT REPORT FOR CS352

Presented by
Kunal Gupta (150001015)
Bhor Verma (150001005)
Computer Science and Engineering
3rd Year

Under the Guidance of
Dr. Somnath Dey

Department of Computer Science and Engineering
Indian Institute of Technology Indore
Spring 2018

CONTENTS

INTRODUCTION

The project is a Football Penalty Shooter Game. The game is similar to that of the Football Penalties in real life. The player is able to interact with the game in several ways. Namely the player can:

- Move around the camera with the mouse/touchpad to get a better field of view.
- Zoom in/out using the keyboard, again to get a better field of view.
- Set the angle of the shot, and thus point it in the desired direction.
- Set the power of the shot, that is, the speed of the ball.
- View the goal statistics and switch from one mode to the other using the keyboard.

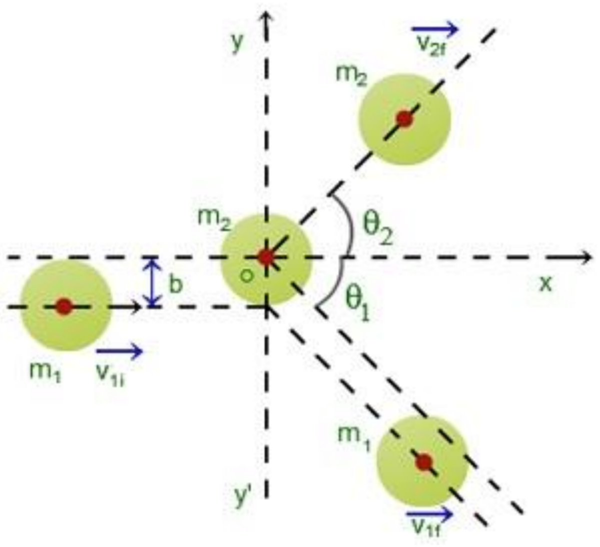
The goal is defended by a moving Defender, which moves in direction of the shot. He will deflect the ball on contact. Points are awarded whether the shot is a goal.

MODULES

THE PHYSICS

This will consist of the physics and the trajectory tracer when the ball is shot.

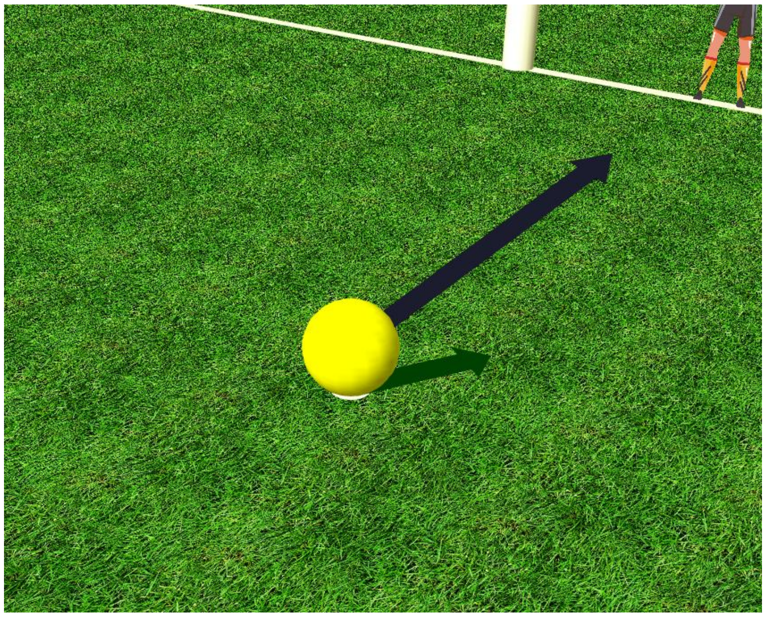
It applies gravity effects, frictional effects as well as controls the bouncing of the ball. It also looks after the collisions of the ball with the defender as well as the poles of the Goal Post, which uses oblique collisions, similar to:



The related calculations are done in the code and are omitted here for simplicity.

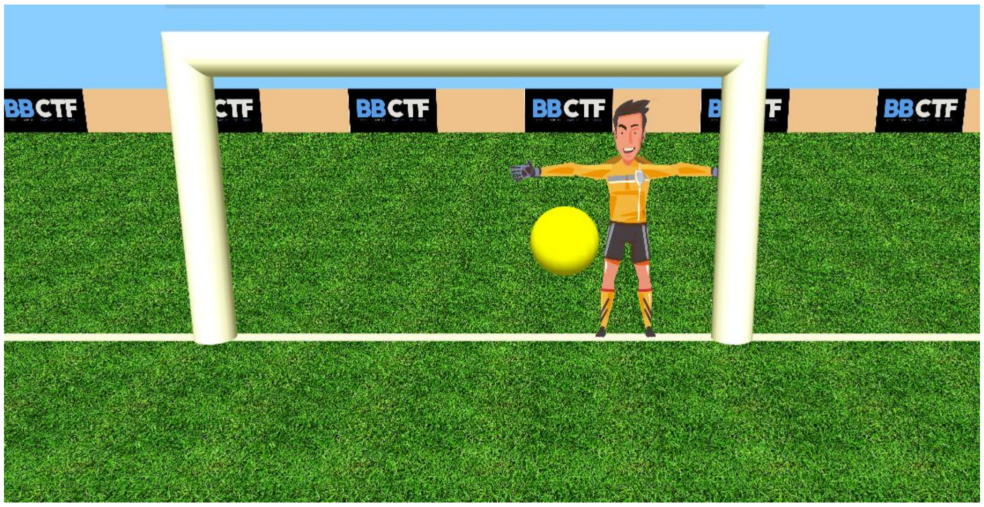
THE SHOOTER

This will consist of the shooting power and angle which the player can set. The angles can be adjusted to a range of -60° to 60° horizontally and from 0° to 50° vertically.



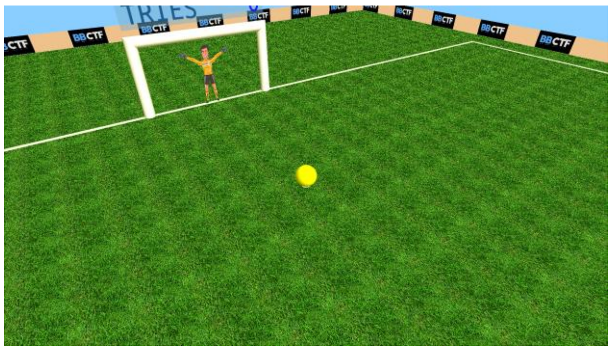
THE DEFENDER

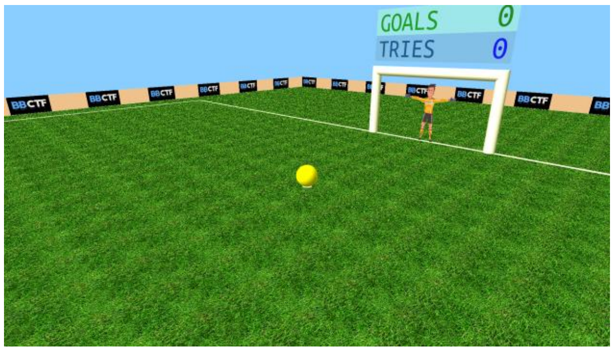
Similar to a real-life goalkeeper, it moves to defend the shot. The hands of the defender move continuously, whereas the whole defender moves in the direction of the shot to defend it.



CAMERA ANGLES

The player can change the camera angles for the penalty using the mouse so as to view the shot in different angles.

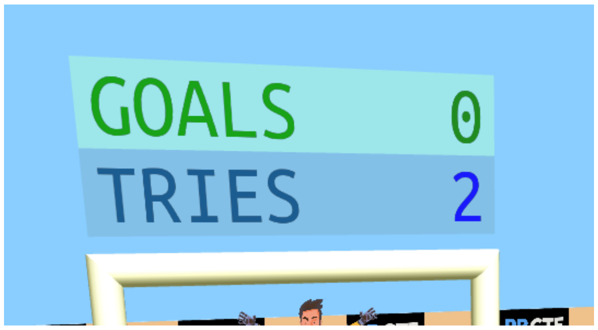


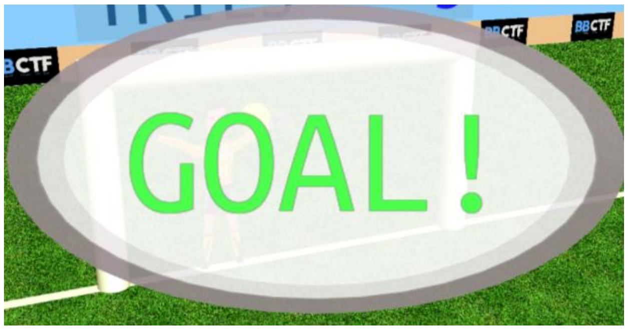


TEXTURE AND FONTS

These contain the required functionalities for loading textures with transparency.

Further we have created a custom font system, which uses transparent textures to use them as font, which can be used in 3D and can be resized, unlike the Raster Fonts, and also have a good width and thus can be easily seen, unlike the Stroke Fonts.

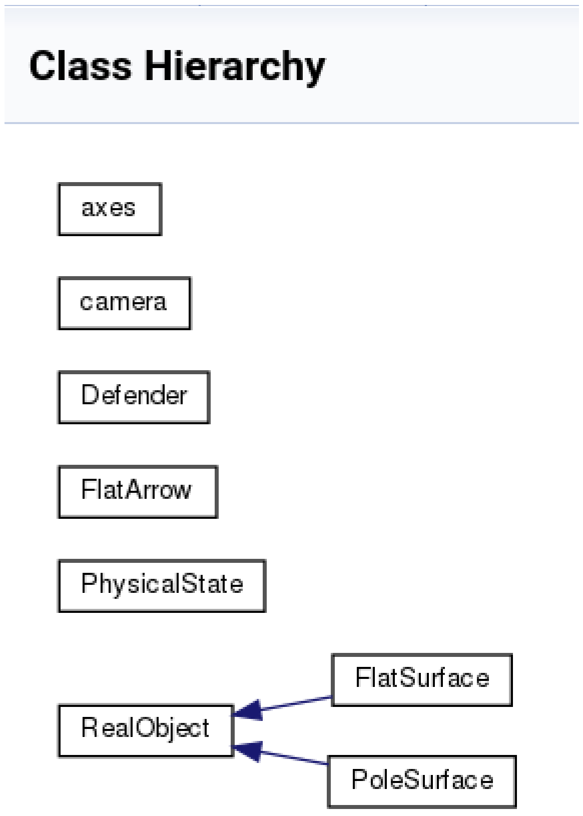




TECHNICALITIES

CLASSES AND STRUCTURES

The classes and structures used in our project broadly have the following relation:



Axes

A helper class for modularity of values of quantities along the 3D axes.

Camera

A class that contains information about the camera: Horizontal and Vertical Angles and distance from the point to be seen.

Defender

Contains the physics of the defender as well the drawing methods.

FlatArrow

The arrow used for setting the shot angle – contains the angles and drawing methods.

Physical State

The helper class that contains the physical state – velocity, acceleration and position – of any object, like the defender and the ball.

Real Object

Used for drawing the FlatSurface (ground) and PoleSurface (poles of the PoleSurface).

FUNCTIONS

functionalities.cpp/.h

- `void handleResize(int w, int h)`
  Handles the window resize using the given width 'w' and height 'h'.
- `double distanceBW(axes axes1, axes axes2)`
  Calculates Euclidean distance of between the given positions in 3D space.
- `bool isItGoal(PhysicalState ball)`
  Calculates whether the given shot is a goal, by inspecting the ball and goal coordinates.
- `void backgroundMusicPlayer(int _)`
  Plays the background music on a loop using the `system()` function and the `paplay` utility in linux.
- `void initialiseEverything()`
  Contains all the methods to be called and the values to be set/reset when the game is started or ball is reset. Is called using the `initialiseEverythingCallback()` function.
- `void initialiseEverythingCallback(int _)`
  A callback function to be used in conjuction with `glutTimerFunc()` to be called every time the screen needs to be reset, for the next or the first shot.
- `void drawGoalPost()`
  Draws the Goal Post.
- `void cameraPosition(axes point, double distance, double zAngle, double xAngle)`
  Is actually a wrapper for `gluLookAt()` for easier camera manipulation using a point, two angles and the distance from the point.
- `void rainBox(double alpha = 0.7)`
  Is used for drawing the colored power meter scale for adjusting the shot power in `SHOOTING` mode.
- `void drawPowerMeter()`
  Calls the `rainBox()` function and draws the arrow corresponding to the power level.
- `void drawHUD()`
  Function responsible for drawing the Heads Up Display or the HUD. It shows the initial Instructions to the game and the power meter.

- `void updateDefenderPosition(int _)`
  Uses the physics responsible to change the position/velocity of the defender stored in the `PhysicalState` class.
- `int convertToTexture(const char *filename)`
  Load the file with the given filename. The file must contain integers to pixel values for each location in the texture. To generate such a file use `textureloader.py` along with a RGBA PNG file. This loaded file is written into a new file with binary data so that it can be loaded faster into an array to be used as a texture.
- `GLuint loadTextureFile(const char *filename)`
  Loads the file created by `convertToTexture()` and creates an OpenGL RGBA texture using it. The returned value is the Texture integer.
- `void drawChalkLines()`
  Draws the white lines on the ground marking the outsides.
- `GLuint convertAndLoadTexture(const char *filename)`
  A wrapper function for calling `convertToTexture()` and `loadTextureFile()`.
- `void start2DTexture(GLuint texture, bool lightingDisabled)`
  Contains the initializations to be done before after after loading the texture. It basically makes the texture ready for use.
- `void end2DTexture(bool lightingDisabled)`
  Contains the clean up to be done after drawing with the texture has finished.
- `float writeMultiLineText(string text, int texture, alignment align)`
  Repeatedly calls `writeText()` for every line of text provided in the given string.
- `float writeText(string text, int texture, alignment align)`
  This is the function created for using textures as fonts. Given a font texture file (can be created using `fontloader.py` and a TrueType font) it crops out each character from the texture and draws it on a transparent quadrilateral and repeats the same for each character, giving it the effect of using another font. Thus we implemented our functionality of loading TrueType fonts.
- `void rotateMsg(int _)`
  A simple callback function used to rotate the 'GOAL!' or 'MISS!' message for a 3D animation, drawn using the `showMsg()` function.
- `void showMsg()`
  Is used for drawing the 'GOAL!' or 'MISS!' message.

main.cpp

- `void showScore()`
  Shows the number of scored goals along with the total shots above the Goal Post.

- `void updatePos(PhysicalState &p, double t)`
  Calculates the Trajectory of the ball, using the friction, bouncing effects and collision and updates the `PhysicalState` of the ball accordingly.
- `void updatePosCallBack(int _)`
  A call back to update the `PhysicalState` of the ball when SHOOTING mode is set.
- `void draw()`
  The function that is passed in `glutDisplayFunc()`. It calls all the modular drawing function in a sequential manner so as to draw the game canvas.
- `void incrementPowerMeter(int _)`
  Increments the power when the SPACE BAR is held down.
- `void idle()`
  The function passed to `glutIdleFunc()` for perform background processing tasks or continuous animation when window system events are not being received.
- `void handleUpKeypress(unsigned char key, int x, int y)`
  Handling key releases on the keyboard. Changes the state of the respective key in downKeys[].
- `void handleSpecialKeypress(int key, int x, int y)`
  Handling key presses of special keys on the keyboard – namely the up, left, right, down arrow keys, to be used in setting the shot angle.
- `void handlePassiveMouse(int x, int y)`
  Handles the mouse pointer movement for adjusting the camera angle for the current view and angle of the game.
- `void myInit(void)`
  Contains initializations such as enabling lighting, depth, blending, anti-aliasing, etc.

shapes.cpp

This file contains the implemented functions like `draw()` of the classes previously describes.

PLAYING THE GAME

BUILDING, RUNNING AND UPDATING

The game can be built from sources using the following commands:

```
$ git clone https://github.com/iamKunal/OpenGL-3D-Football-
  Penalty-Shooter.git      #Latest on GitHub
$ cd OpenGL-3D-Football-Penalty-Shooter
$ mkdir build && cd build
$ cmake ../
$ make
```

The game can be run by running the following the build directory previously created.

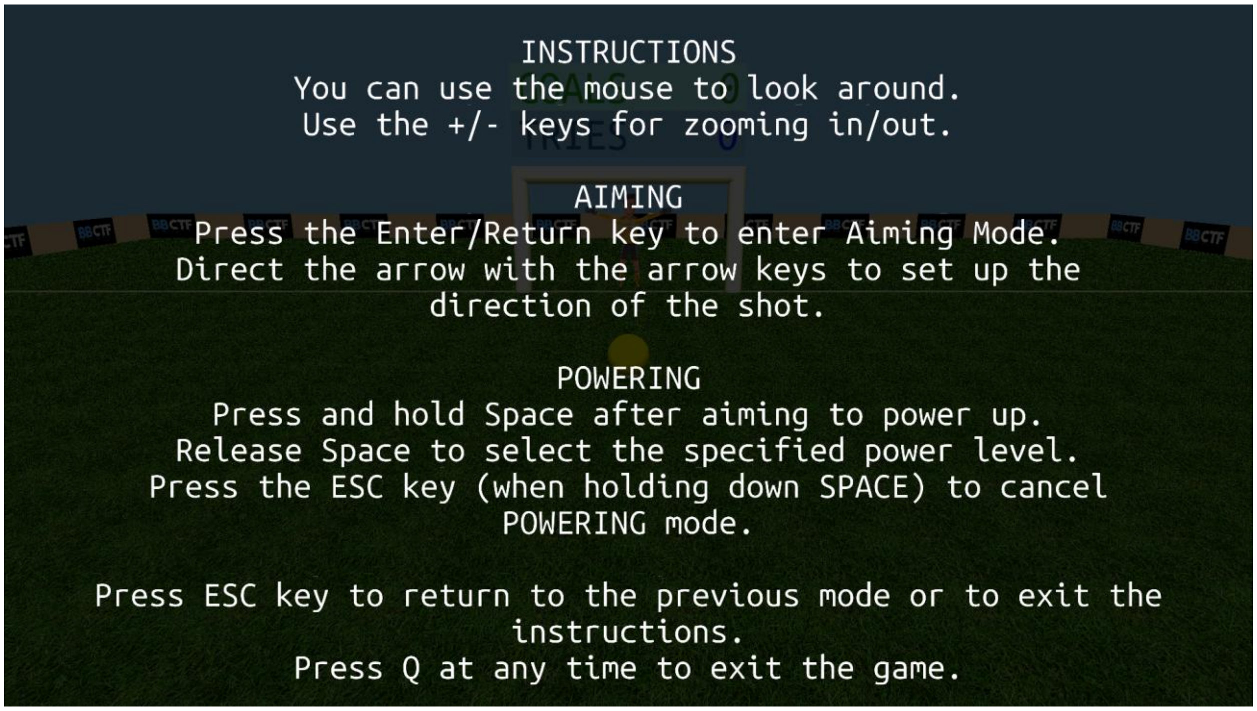
```
$ make run
```

To update the game simply run the following from the root directory of the game.

```
$ git pull
```

PLAYING

Playing the game is simple as the game also provides the instructions for playing:



```
INSTRUCTIONS
You can use the mouse to look around.
Use the +/- keys for zooming in/out.

AIMING
Press the Enter/Return key to enter Aiming Mode.
Direct the arrow with the arrow keys to set up the
direction of the shot.

POWERING
Press and hold Space after aiming to power up.
Release Space to select the specified power level.
Press the ESC key (when holding down SPACE) to cancel
POWERING mode.

Press ESC key to return to the previous mode or to exit the
instructions.
Press Q at any time to exit the game.
```

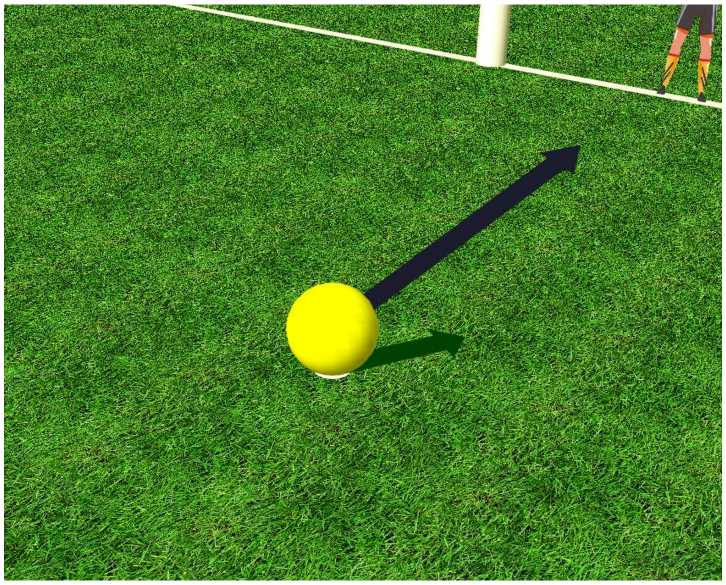
This mode is also called the HELP mode internally.

Adjusting Mode

The initial screen after quitting the instructions is internally called ADJUSTING mode and is used for just looking around in the game with the mouse.

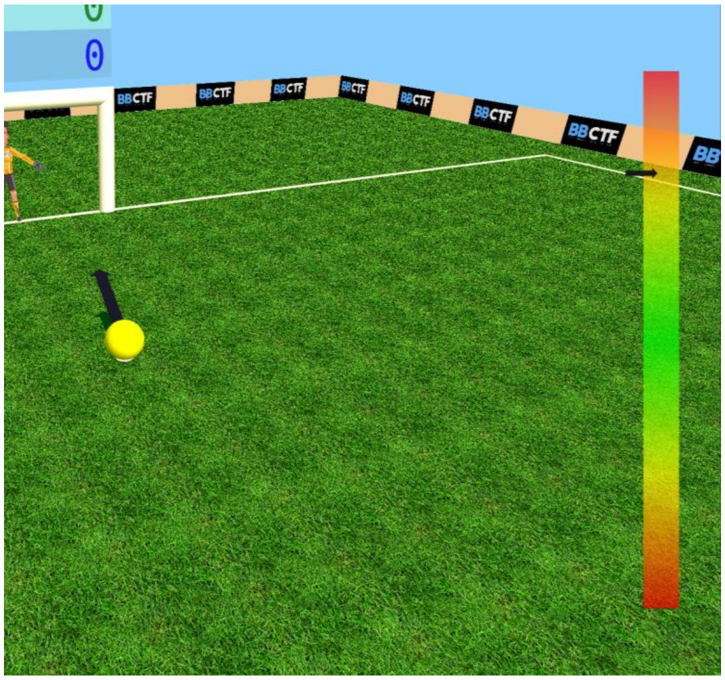
Aiming Mode

The player can change the angle of the shot (same as the angle of the blue arrow) using the arrow keys on the keyboard.



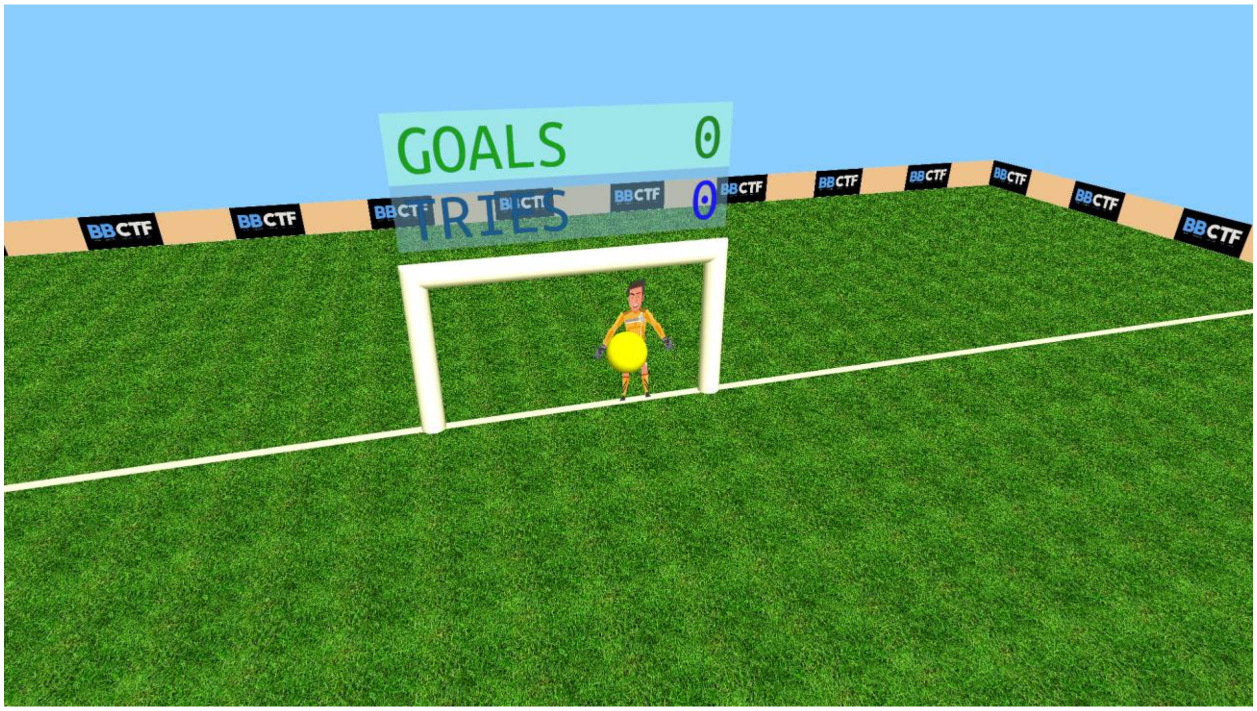
Powering Mode

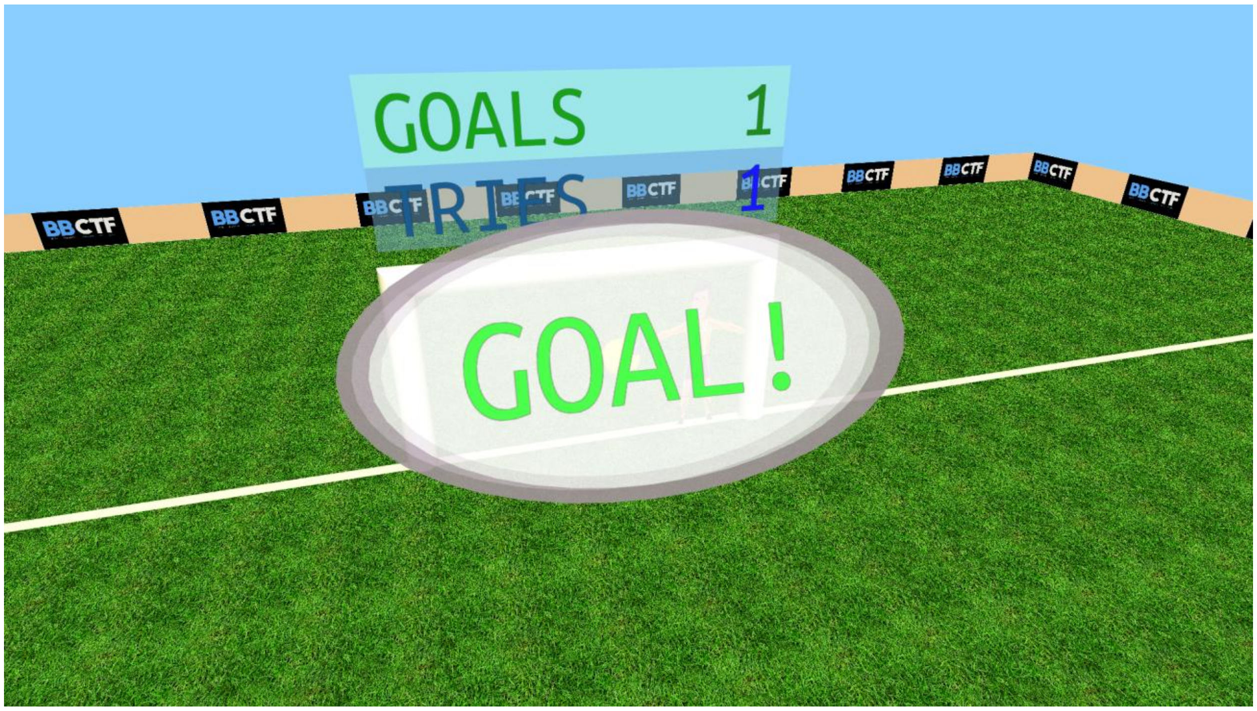
Herein, the player can press and hold the spacebar for the desired power which can be seen using the arrow on the power meter and then release it at the desired power level.



Shooting Mode

Internal to the game, the shooting mode is the actual mode where we can see the shot in action.


GOALS 0
TRIES 0
BBCTF


GOALS 1
TRIES 1
GOAL!
BBCTF