**3D Space Shooter Game**

**Software Architecture Document**

**Game Description:**

My game is a 3D space shooting practice game. In this game, the player will try to aim at spheres that are random generated. The player can move freely in 3D space, and can also move his vision to the left, right, up, or down(pitch & yaw) --- The movement is acceleration-based since it’s in space where there is no friction, therefore each consecutive key press accelerates the vehicle in a certain direction. Furthermore, there’s an easter egg in the game! Press Q and E to rotate the screen and do a “barrel roll”! There is a red cross hair in the center of the screen, and when that crosshair matches a target, it turns green indicating the target is locked. To shoot, simply press the spacebar and if it succeeds, the sphere will disappear. Each successful shot is worth 1 point. The goal is to shoot as many spheres as you can.

**Timed vs Hardcore Level:**

The player can choose between 2 modes in the main menu UI.

In both modes, the game initializes with 30 spheres and a new sphere is generated every 2 seconds.

Timed Mode: The player has 2 minutes to shoot as many spheres as possible.

Hardcore mode: The player loses when the amount of spheres reaches 35, meaning he must shoot spheres faster than they can regenerate.

**User Interface(UI):**

In the beginning, the player sees a mainmenu window with a highscore displayed on top, a welcome text, and 4 buttons -> Timed, and Hardcore, quit game, and rules.

As soon as the player chooses a mode, the game begins.

To gain points, the player must aim the crosshair at the randomly generated spheres on the screen.

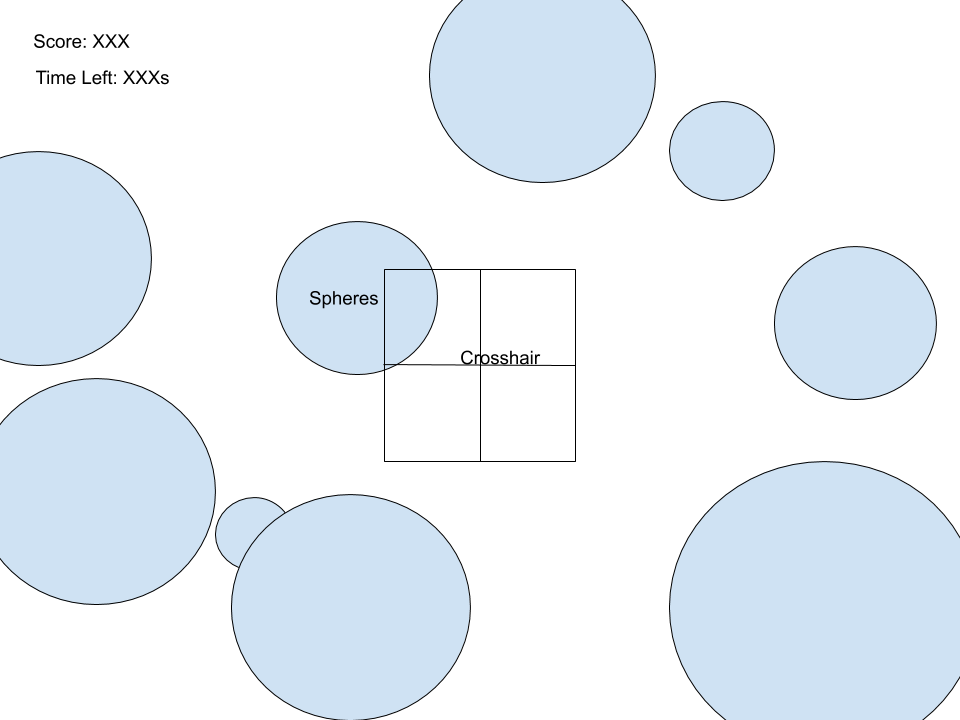
When the game ends, the player gets taken to an endgame screen: It will display “Gameover!”

There will be a button called Exit to exit back to the main menu.

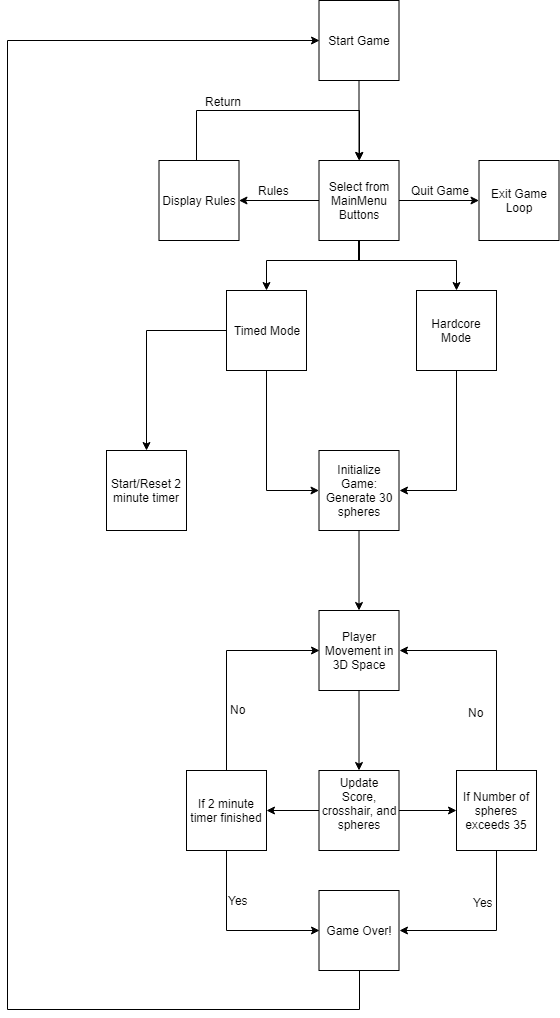
**Rules Menu (printed for the player):**

* Timed mode: Your goal is to shoot down as many spheres as possible within 2 minutes
* Hardcore mode: Your goal is keep the number of spheres to be less than 35.
* The game initializes with 30 spheres and a new sphere is generated every 2 seconds.
* Controls:
* To aim up, press W; to aim down, press S
* To aim left press A; to aim right, press D
* To go forward press F, to go back press G
* Press [+] to aim harder, and press [-] to aim less
* To shoot, press the spacebar.
* Good luck!

**Screen Sketch:**



**UML Diagram**

****

**Schedules & Goals to Achieve**

1. **Software Architecture Document**
   1. Due date: May 13, 2019
2. **Oral Progress Report:**
   1. Due date: May 16, 2019
   2. We will complete the oral progress report by May 16
3. **Program**
   1. Due date: May 26, 2019
   2. We already have started programming and we hope to complete the program by 25th(5 classes)
      1. See bottom of page for more specific goals
4. **Written Report**
   1. Due date: May 29, 2019
   2. We will complete the written report by May 28
5. **Presentations**
   1. We will prepare for the presentations after May 31st

**Programming Challenges(Complete all by May 25th)**

* **Basic Game Programming**
  + Create a main class
  + Create a sphere class(object)
  + Create a renderUtils class
  + Create a player class
  + Create main game loop & initialization variables…
  + etc...
* **3D to 2D projection**
  + Figure out the math(linear algebra) formulas for 3D to 2D conversion
  + Create to2D() function in RenderUtils
* **3D Translation**
  + Figure out the math(linear algebra) formulas for translating object in 3D space + compatibility with to2D() function.
  + Add 3D translation features within to2D() function using playerX, playerY, and playerZ
* **3D Rotation**
  + Figure out 3D rotation matrices and the math(linear algebra) required to rotate points around the position of the player
  + Create transformPoints() function
  + Add rotation to points in to2D() prior to converting to 2D
* **3D Sphere scaling**
  + Figure out math to scale the 3D sphere according to how far away the player is from the sphere
  + Integrate math into drawPoints function
* **Bullet and Sphere Collision**
  + Figure out Math required to detect whether or not the crosshair is aiming directly at a sphere
  + Find a way to delete that specific sphere from the array containing all sphere objects
  + Integrate math into Sphere and RenderUtils class
* **Check if object is behind player**
  + One problem with the to2D function is it projects objects behind and in front of the player, therefore showing objects that shouldn’t be showing
  + Challenge: Since the coordinate system allows the player to freely move around and look around in 3D space, the math isn’t as simple as checking if(sphere.z<player.z) since when the player’s yaw and pitch is changed, to find whether an object is behind the player, you must check if it’s behind in respect to the plane at which is rotated according to pitch and yaw: This magnifies such a simple problem to an extremely difficult linear algebra problem
  + Solve the above problem with linear algebra
  + Integrate math into to2D() function.
* **Userinterface(UI)**
  + Create Main Menu UI
    - 4 Buttons: Quit game, Rules, Timed Mode, Harcore mode
    - Create functions for each button to do their tasks
  + Create Rules Menu
  + Create Game Over Menu
* **Game Logic**
  + Figure out functions for:
    - 2 Minute timer
    - Jump to game over menu when player loses the game
    - Initializing game
    - Random sphere generation