|  |
| --- |
| -White Star Studios |
| Project Proposal |
| Hungry Yoshi |

|  |
| --- |
| Ori Talmor  5/22/2015 |

# Purpose

* A video game controlled by the Kinect where the player has to slingshot a hungry Yoshi character towards its apples.
* The player has to project the Yoshi character through trampolines and other obstacles without dying until it reaches its target. At which point the level is complete and the following level is loaded.
* A 2D puzzle platform game which highlights:
  + Multiple Levels
  + The ability to create your own levels
  + Kinect Integration
  + Engaging puzzles
  + A creative twist of Angry Birds, Cut the Rope, and Base Jumpers
* This assignment will demonstrate my grasp of Object-Oriented-Programming and the design process.
* It will teach me new skills like rotated rectangular collision, working with Kinect, and projectile motion.
* Will test my programming and problem solving skills and capabilities.

# Objectives

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Objective Name** | **In-Software Description (What and Where)** | **Owner** | **Difficulty (1 to 10)** |
| 1 | Kinect Integration | * Set up the main controller for the game to work from the Xbox 360 Kinect. * The slingshot mechanism that determines the players’ trajectory * Captures audio and performs operations * E.g. when the player says “Shoot” the character will shoot out of the slingshot. | Ori | 6 |
| **Technical Description:**   * Map the joints from the Kinect. Calculate distance from wrist joints, used for velocity. Use angle from left hand to right hand as the trajectory angle and voice recognition to shoot the player. * To continue to the next level, the hand joint needs to go over the next button and needs to be held there for a few seconds. The user controls the game from the Kinect. | | | | |
| 2 | Collision Detection | * Make sure the player stays within boundaries * Detects when the player collides with objects in the environment * Circle to circle collision detection as an indicator for potential rotated rectangular collision. * Rotated rectangular collision detection between character and almost all objects in the map (trampolines, spikes, bubbles, etc.). | Ori | 10 |
| **Technical Description:**   * For the boundaries, perform basic collision detection, to make sure that the character does not go off-screen. * For everything else create a bounding circle around each object with the diameter set to either the length or width, depending on which one is larger. Then check to see if the distance between the centers of any two objects is less than their added radius’. This would mean there is an intersection/collision. * If the above collision detection passes, then check for rotated rectangular collision by:   + Calculate the perpendicular axis’s to every side of the rectangles.   + Project the vectors representing the four corners of each rectangle onto each of the axes.     - Equation:   + Find the Maximum and Minimum projected value for every rectangle on each axis.   + Add the two values of each point together (X & Y) in this equation:     - A.UR.X \* Axis1.X + A.UR.Y \* Axis1.Y   + With every one of the above scalar values, check: | | | | |
| 3 | Level Designer | * Gives the player the ability to create his/her own levels and then play them * Will be made in game with the Kinect controller or the mouse and keyboard and will then be saved in the program directory. * Will be loaded in game like every other map and the player will then be able to play the level with the same objective | Ori | 6 |
| **Technical Description:**   * Display to the screen the map and all the available tiles that the player can add. Force the player to add an end tile, but also create a permanent start tile. * The player can create the map in two main methods:   + Kinect     - The player can grab objects by holding their hands over the tile in the menu that they want to add.     - To place a tile on the map, the player, with the picked up tile, holds their hand over the location that they want to place it.   + Mouse and Keyboard:     - The player can click on the tile that they want from the menu, and then hover over the location and then click again * When the player finishes adding all the tiles they want and press the create button, the map is created and stored a text file for future loading. | | | | |
| 4 | Physics Engine | * Projectile Motion   + Affect how the Yoshi moves once it is released from the slingshot   + Will be solely responsible for the players’ movement once it has the data from the slingshot * Acceleration & Deceleration   + The objects in the environment change the speed of the player and affect its trajectory. * Different objects affect different physics in the game. Gravity, speed, Acceleration, Deceleration… etc. | Ori | 7 |
| **Technical Description:**   * Calculate the trajectory of the player by using the following equations: sinθ = Vy/Vi or cosθ = Vx/Vi then move the players’ position by that amount over time based on the angle and initial velocity. * Given the angle and the initial velocity calculated by multiplying the distance of the slingshot by a factor, the x and y velocity of the character is calculated and is used to update the motion of the character across the map. * The destination is calculated for the trajectory and the character rotated towards that destination using the Math.Atan2() method. * By hitting objects such as the trampoline on the path, the direction is reversed, and the destination recalculated based on the velocities upon impact. * When hitting objects like the bubble, the player quickly decelerates and begins floating. Objects like the fan reaccelerate the player. Acceleration is calculated using the formula a = v / t. This acceleration is added on to the velocity of the character at every update. | | | | |
| 5 | Particle System | * Effects that will appear when the character hits certain objects (like a wall or spike) as well as fly through the air. * Particles of objects/shapes of different sizes, angles and transparency that will appear at different times to look like particles emitted from the character. | Ori | 7 |
| **Technical Description:**   * Create the particle object which holds the texture, position, velocity, angle, angular velocity, color, size, and the particles’ living time.   + This object updates the particles’ position and angle by the angular velocity, manages its living time, and it also draws the particles. * Create a Particle engine class to hold the emitting location, a list of particles, and the list of textures.   + The engine generates the particles by choosing a random texture, and a random velocity, color, angular velocity, size, and lifetime within a range. This class also draws the particles as well as adds new ones every update and removes “dead” particles. | | | | |

# Milestones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Task #** | **Description** | **Owner** | **Due Date** | **Dependent Task #s** |
| 1 | Set-up Kinect Manager and map needed joints (Objective 1) | O. Talmor | May 27 | None |
| 2 | Scale Kinect movements and audio recognition  (Objective 1) | O. Talmor | May 28 | 1 |
| 3 | Create Map and Tiles | O. Talmor | May 29 | None |
| 4 | Load and Save File | O. Talmor | May 29 | None |
| 5 | Create Selection and Game States controller tool in Kinect  (Objective 1) | O. Talmor | May 30 | 1,2 |
| **6** | **Objective 1: Kinect Manager** | **O. Talmor** | **May 30** | **1,2,5** |
| 7 | Projectile Motion and Trajectory (Objective 4) | O. Talmor | May 31 | 6 |
| 8 | Rotation of Character (Objective 4) | O. Talmor | May 31 | 7 |
| 9 | Game Boundary Collision Detection (Objective 2) | O. Talmor | May 31 | 7 |
| 10 | Circle – Circle Collision (Objective 2) | O. Talmor | June 1 | None |
| 11 | Calculate for Corners of Rectangle (Objective 2) | O. Talmor | June 2 | None |
| 12 | Projection onto each axis (Objective 2) | O. Talmor | June 3 | None |
| 13 | Test collision with projected vector locations (Objective 2) | O. Talmor | June 3 | 8 |
| **14** | **Objective 2: Collision Detection** | **O. Talmor** | **June 3** | **9 🡪 13** |
| **15** | **Milestone 1: User Input and Collision Detection** | **O. Talmor** | **June 3** | **1 🡪 14** |
| 16 | Camera and Zoom Capabilities | O. Talmor | June 4 | 7 |
| 17 | Create Particle System Object (Objective 3) | O. Talmor | June 5 | None |
| 18 | Implement Fountain Particle System in player trajectory (Objective 3) | O. Talmor | June 5 | 17 |
| **19** | **Objective 3: Particle System** | **O. Talmor** | **June 5** | **17,18** |
| 20 | Acceleration and Deceleration system  (includes bubble object)(Objective 4) | O. Talmor | June 6 | 7 |
| **21** | **Objective 4: Physics System** | **O. Talmor** | **June 6** | **7,8,20** |
| 22 | **Milestone 2: Physics and Particle System** | **O. Talmor** | **June 6** | **7,8,17,**  **18 🡪 21** |
| 23 | Level Designer Display(Objective 5) | O. Talmor | June 8 | 1-5 |
| 24 | Level Designer Save Map(Objective 5) | O. Talmor | June 8 | 3,4,23 |
| **25** | **Objective 5: Level Designer** | **O. Talmor** | **June 8** | **23,24** |
| 26 | Menu | O. Talmor | June 9 | None |
| 27 | Bug Testing | O. Talmor | June 10 | 1 🡪 26 |
| 28 | **Milestone 3: Level Designer and Final Touchups** | O. Talmor | June 10 | 1🡪 27 |

**Declaration:**

**I have read the statements regarding cheating in the assignment description. I affirm with my signature that I have worked out my own solution to this assignment, and the code I am handing in is my own.**

**Signature: Ori Talmor**