CSC8503 Coursework Marks

**Coursework Part A:**

**You are to make a game in which the player must aim to get their ball to the end of the game in the fastest time possible. Once the game finishes, the player should be shown their final score, and time taken to complete the game. The player does not control the ball directly, but instead may click on objects within the world to trigger actions within the world (tiles that bounce the ball upwards, or platforms that rotate to ‘flick’ the ball through the world, for instance). The game should feature a simple menu, allowing the player to play again, or exit the game.**

**The program you create should take place within a single virtual environment - the exact composition of this is up to you, but it should have obstacles placed around (these can consist of simple moving and rotating cuboids and spheres) to make the world more interesting. The level should also have surfaces that produce different effects when the ball moves on them (jelly could make the ball bounce, or ice could make it hard to change direction).**

**If the player clicks on any object within the world, the game should display some debug information (using the provided text rendering code). This could consist of position and orientation, or information on the current state of any state machines or AI used within the world.**

**Coursework Part B: The Evil Maze and Other Terrors**

**The game should be extended with an additional game mode, accessible via the main menu. In this mode, an additional challenge is present in the game world – the Evil Maze. This should take the form of a maze comprised of a floor and walls made of cubes. The ball can be moved as you wish – launched via specifically placed springs, or a tilting floor, for instance. An additional challenge is contained within the maze – when the player enters, an enemy ball should spawn in one of the corners of the maze. This enemy ball should use appropriate pathfinding methods to move towards the player’s ball – if they come into contact, both the enemy and player balls respawn at their respective starting positions.  Powerups should also spawn in the maze (they could freeze the enemy ball, provide bonus points, or some other gameplay effect of your choosing); the enemy ball should use appropriate AI methods to make decisions on whether to chase after the player, or try to collect any nearby powerups.**

|  |  |
| --- | --- |
| **Part A** | **Checklist** |
| application of forces |  |
| simple state based logic used in the challenges |  |
| allow the player to move the ball via the application of forces |  |
| the program successfully detecting and resolving collisions with the environment (AABB, Sphere, OBB, plane) (impulse, projection, or penalty, with gameplay effects) |  |
| apply gameplay effects from the collection of items |  |
| Debug information on selected object, position, orientation, state |  |
| Surfaces with different effects (jelly, ice) |  |
| On game finish, display time and score |  |

|  |  |
| --- | --- |
| **Part B** | **Checklist** |
| collision detection |  |
| an enemy ball within the maze that uses state machine logic |  |
| AI opponents should use appropriate pathfinding to navigate through the world towards the goal |  |
| Constraints should be used to create more interesting obstacles |  |
| Appropriate logic should be used for the menus |  |

TO DO:

Add a way for decision making based on collision detection eg, ball and finish or ball and powerup – maybe use onCollisionBegin. Currently added in a print statement but its in wrong place