James Burrell 100263300

CMP-5010B Reassessment Report

2) Motion and control of the user controlled character(s) and/or objects.

The placement and movement for the character and objects in the game is based on a coordinate system that has the same aspect ratio as the window. The height and width of the window is set as a constant at the beginning of the program, and the viewport is set as a multiple of the screen height and width. This ensures the objects in the game maintain their aspect ratio if the window is resized. This leaves the coordinate system as 60 to -60 in the X axis, and 30 to -30 in the Y axis. As all objects in my game are rectangular, their 4 vertices are defined using minimum and maximum values of X and Y.

Assuming no collisions will happen, when the the left or right arrow keys are pressed the minimum and maximum values of X of the player are increased at a small, constant rate to simulate a uniform speed in the horizontal direction and smooth movement.

As for the vertical direction, again assuming no collisions, the up arrow causes the player to enter a jump and a corresponding boolean is set to true. Its minimum and maximum values of Y are increased until a fixed point above its starting position is reached, using a jump height that is defined in the player class. Once this point has been reached, the player is considered to have finished jumping, and the boolean is set to false to indicate this. Whenever the player is not jumping, a method is called every frame to simulate gravity and pull the player in the negative X direction. Similarly a boolean is set to true whenever the player is falling which works in order to ensure the player cannot jump mid-fall. The gravity also functions to tell the jumping method when the player has landed on an object, so the starting point for the next jump is set correctly.

As with movement in the X direction, Y movement is incrementally increased/ decreased at a constant rate each frame and so moves at a fixed speed when jumping and falling. A more realistic ‘arching’ jump could be implemented using a velocity vector and a way of recording the change in time since the jump was started, in order to reduce the velocity vector as it approaches the end of the jump duration. This process could also be adapted to make gravity in the game more realistic, decreasing the velocity vector at a changing rate until a fixed ‘terminal velocity’ is reached.

3) Collision detection & Response between the objects/ characters in the environment.

The collision detection and response in my game is basic. I have a single collision method that uses axis-aligned bounding box maths to check if two rectangles passed in (in the form of minimum and maximum coordinates) overlap in any way. When the key to move the player in the left or right direction is pressed, a bounding box that represents the player's next position is passed to the collision method and if it overlaps, the minimum and maximum value of X is left unchanged so the player does not move when the next frame is displayed. Movement in the up and down direction is also checked for collisions in this way. When jumping only upwards

collisions are checked for, and then when falling only downwards collisions are checked for, and movement stopped in the relevant way.

The upside of checking for collisions only when the movement keys are pressed and only in the direction of movement, is a reduction in the number of collisions that have to be checked, which increases game performance. However because of how simple the game is this performance increase is unnoticed. However, if the game was more complex with more entities to check collisions against, it would increase performance. Another benefit of this process is that it allows the player to move in the positive or negative X direction even when colliding with a platform from above or below. If all directions were checked at once, the player would get stuck as soon as it touched a platform or the ground, as its next position would be considered a collision.