Hashmaryne van Cuylenburg

UCLan Id- G20761887

CollectIt! 3D game documentation

CO1301: GAMES CONCEPTS- ASSIGNMENT

Contents

[1. Overview 2](#_Toc516867587)

[2. Gameplay 3](#_Toc516867588)

[3. Design 4](#_Toc516867589)

[4. Implementation 6](#_Toc516867590)

[4.1. Loading models and camera 6](#_Toc516867591)

[4.2. Orb Movements 8](#_Toc516867592)

[4.3. Collison detections 10](#_Toc516867593)

[4.4. Text Output 13](#_Toc516867594)

[5. Conclusion 14](#_Toc516867595)

[5.1. Challenges faced 14](#_Toc516867596)

[5.2. Further Enhancements 15](#_Toc516867597)

# Overview

CollectIt! 3D is a game loosely derived from the popular video game concept, “Snake”. CollectIt! is a desktop game which is developed using the C++ programming language and the gaming engine; TL engine. The main objective of this game is to score as many points as possible by collecting orbs which are found in random positions while avoid obstacles (example- boundaries and poisonous orbs)

I had initially set out to develop a 3D version of snake but due to challenges faced when finding models and implementing certain common traits in snake games, I decided to come up with my own game which is loosely based on snake.

CollectIt! is not only a fun and addictive game to play, but its also fairly simple to develop.

The game is currently quite basic, but I plan on improving the game further by improving graphics and making it more challenging.



# Gameplay

The CollectIt! gameplay is fairly simple. In this game, the player has to collect the red orbs which appear in random positions using the yellow orb. The collected red orbs will be added to the back of the yellow orb. (This is similar to how the snake’s size increases when food eaten, in a snake game). 10 points will be added to the score every time a red orb is collected. If the player collects a blue orb, 10 points will be deducted. The orbs can be collected by only the yellow orb.

The challenge of the game is to avoid obstacles. There are 3 white orbs that act as obstacles. They move horizontally within the boundaries. If the yellow orb or any of the collected red orbs collides with any of the white orbs, the game will be over. Therefore, the more red orbs that are collected, the more challenging the game would be as it opens up more possibilities of colliding with the white orbs. Moreover, if the player goes out of the bounds with the yellow orb, the game will be over.



# Design

This game has a fixed camera view which shows a landscape representing a ground in which the orbs are placed. The white orbs will be moving horizontally across the ground. The red and blue orbs are placed in random positions. At the center above the ground, the name of the game (CollectIt! 3D) is shown. Instructions on how to play the game are shown on the top left corner of the screen. The controls for the game are shown at the top right corner of the screen. The score is shown on the top center of the screen.



1. Instructions.
2. Score.
3. Game title.
4. Controls.
5. Red sphere which gives 10 points
6. Whites spheres that move horizontally.
7. Yellow sphere which collects red spheres.
8. Collected red spheres.
9. Blue sphere which reduces 10 points.

Once the game is over, the following screen is shown,

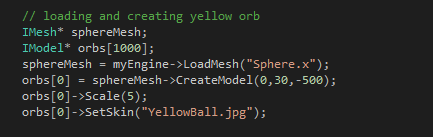


# Implementation

To implement this game, C++ is used along with the UCLan game development engine; TL Engine. All the models used in this game are taken from the TL Engine media folder.

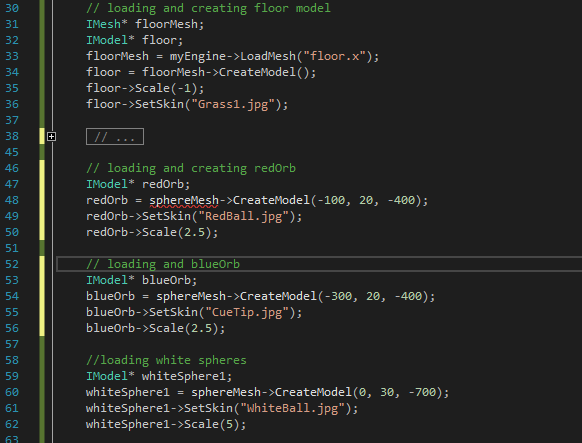
## Loading models and camera

The following code is used to load the orb models,

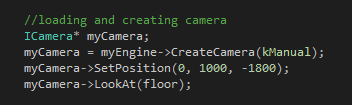


A mesh of type IMesh is first loaded using the “Sphere.x” model called “sphereMesh”. Then an array of type IModel called “orbs” is created. A model is then created using the “sphereMesh” and is assigned to the 0th index of the “orbs” array. This will be the main orb. An array is used here instead of a variable because there will be more orbs that needed to be created when the red spheres are collected. The skin of the model is set to “YellowBall.jpg” and the model is scaled up by 5 in order for it to be more visible.

Other models such as the floor, red, blue and white orbs are loaded in the same way except they’re stored in a variable and not an array,

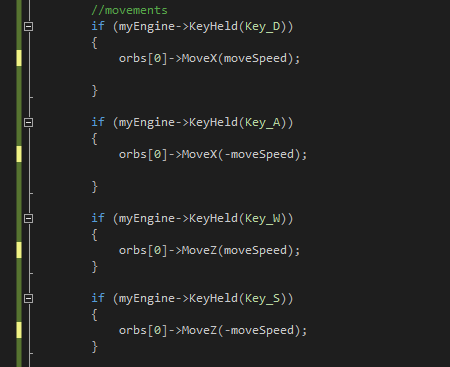


The following code loads and creates the camera. The camera type is kManual and has a fixed position,



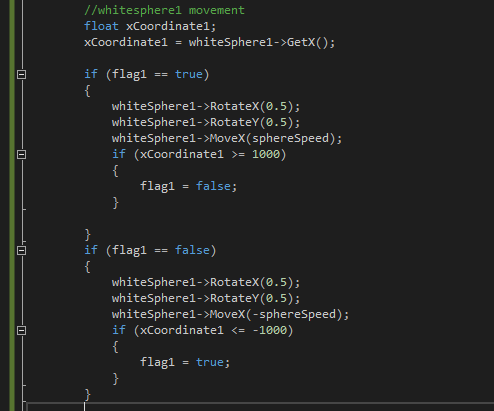
## Orb Movements

To move the yellow orb W, A, S and D is used. W to move forward, A to move left, S to move backward and D to move right, The Key Hit command is used to implement this as seen below,



To move right, the orb is moved along the X axis with the “moveSpeed” (which is 0.5) and to move left, the orb is moved along the X axis with the negative “moveSpeed”. To move forwad, the orb is moved along the Z axis with the “moveSpeed” and to move backward, the orb is moved along the Z axis with the negative “moveSpeed”.

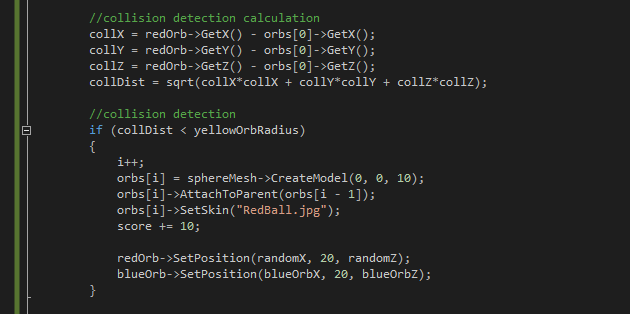
To move the white orbs horizontally, the following code is implemented,



Here, the flag is initialized as true outside the main method. If the flag is true, the orb will move right along the X axis with the “sphereSpeed” (which is 1.5) until the x coordinate is 1000 (this is the right end of the ground.-1000 is the left end). Once the x coordinate is 1000, the flag is switched to false. If the flag is false, the orb will move along the X axis with the negative “sphereSpeed” until the x coordinate is -1000. Once it is -1000, the flag is switched back to true.

## Collison detections

The following code is implemented to calculate and detect collisions between the yellow orb and the red orb. If a collision occurs, an orb model will be created and attached to the yellow orb. 10 will be added to the score. Then the red orb and the blue orb are set in a different random position.



To detect a collision, first the vector between the two models are calculated. Then the distance between the two models will be found by calculating the length of that vector.

The vector is calculated using the following equation,

V(X2 – X1, Y2 – Y1, Z2 – Z1)

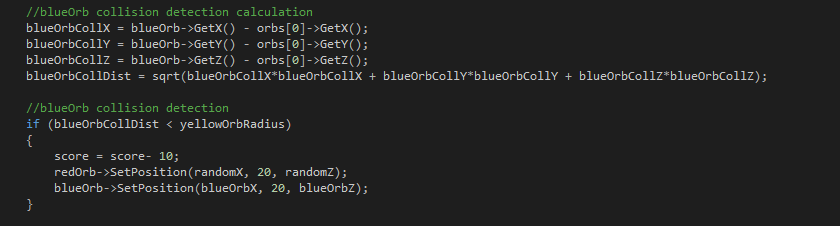
And the length of the vector is calculated using,

Length= 

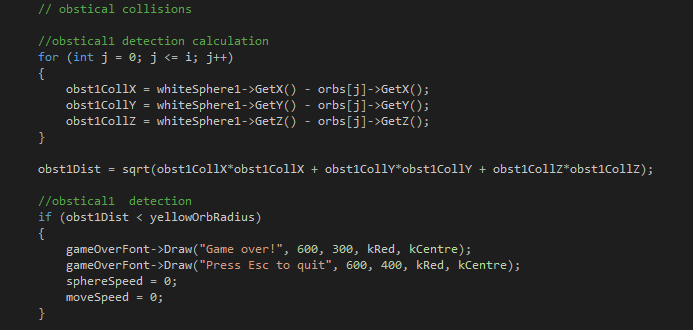
If the distance is less than the bounding sphere radius, then a collision will occur.

The rest of the collisions too are calculated using the above method.

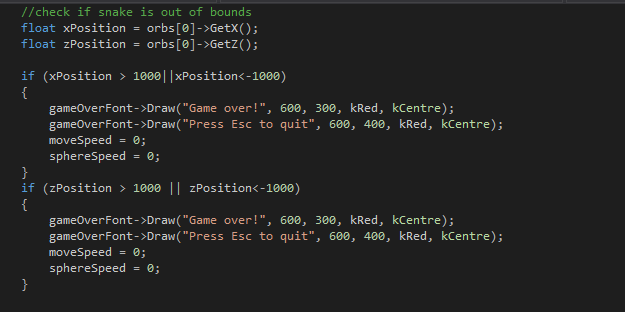
If there is a collision between the yellow orb and the blue orb, 10 will be deducted from the score and then the red orb and the blue orb are set in a different random position.



If there is a collision between any of the collected orbs or the yellow orb and the white orbs, the game over message will be printed and the “sphereSpeed” and “moveSpeed” will be set to 0 in order to make the models stop moving.



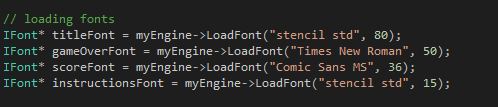
To check if the yellow orb has gone out of bounds, the following code is implemented,



The x and z positions of the orb is gotten by using the GetX and GetZ functions. If the x and z positions are greater than or less than the limit of the floor, then the game is over.

## Text Output

The following code loads the fonts to output text. The first argument is the name of the font and the second argument is the size of the text.

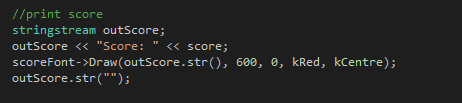


Using these fonts, output text can be printed as shown in the following codes. Arguments taken are the X position, Y position, text colour and text alignment.

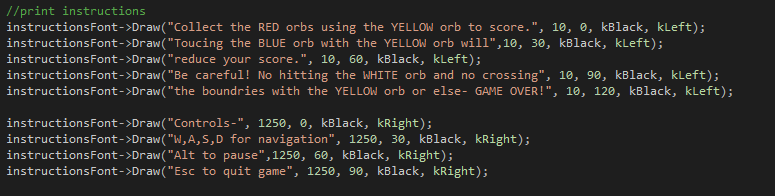
Printing game name:



Printing score: The sstream library is used here.



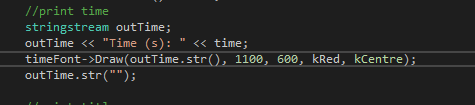
Printing instructions and controls:



Printing game over:



Printing time:



# Conclusion

## Challenges faced

When implementing this game, I came across many challenges. At the start I was set out to make a 3D version of snake. However, I was unable to find .x models of snakes to load on the TL engine. I then attempted to convert other types of models into .x and load them on the TL engine. But this gave me a runtime error. Therefore, I decided to make the snake using spheres that are already in the TL engine media folder. But when coding the game, I was unable to figure out how to bend the snake. And therefore, I came up with the idea for CollectIt! 3D. Due to the challenges I faced, I had a tight time constraint which made it difficult for me to improve the game further.

## Further Enhancements

In the future, I hope to implement the following in order to improve my game,

* Having a menu screen.
* Having different difficulty levels.
* Having sounds.
* Having a high score leaderboard.