Games Engine Construction ICA 2

Ben Shoesmith

Q5122113

Report

**User Guide**

How to Play

The aim of this game is to catch as many chickens as you can by colliding with them. The controls for this game are:

W – Up Movement

A – Left Movement

S – Right Movement

D – Down Movement

Known Bugs and Issues

One issue that comes up is that the background music doesn’t play. I’ve implemented a sound effect for when the player catches a chicken, and that works fine. Also, there currently is no end-game feature, so the game goes on forever.

Implementation Check List

**Adequate Rated Items (D to C)**

**Graphics**

* A ‘black boxed’ graphic system is in place
* Textures can be efficiently drawn to arbitrary positions on and partially off the screen (clipped)
* Animation is implemented and working correctly

**World State**

* A player entity exists
* Input is recognised and can be used to alter the world state e.g. move the player entity
* The Xbox controller is supported and ‘hot pluggable’

**Code Quality**

* Class interfaces are minimal and complete. Class function and member variable visibility is correct
* Code can be built and executed without compiler errors or warnings in debug and release
* Code is well commented
* There are no memory leaks
* There is good error handling throughout
* You have followed all the submission requirements e.g. made a video, submitted the correct files etc.

**Report**

* All requested sections have been attempted adequately and the report is professionally presented

**Good Rated Items (C to B)**

**World State**

* A world model system is in place. It is separate from other code and black boxed
* There is a game loop handling input, world update and rendering
* Bounding rectangle collisions are detected
* There are multiple world entity types

**Code Quality**

* Good use of object oriented techniques e.g. polymorphism, member variable visibility
* Memory is only allocated / deallocated outside of the game loop
* Const is used correctly

**AI**

* Some AI routines are in place e.g. enemy entities move around the world following paths, use state machines etc.

**Report**

* This report would allow another programmer to work with your code systems

**Other**

* Some sound effects are in place

**Excellent Rated Items (B to A)**

**Graphics**

* Interpolation is used to smooth entity movement

**World State**

* The player entity can shoot projectiles (or equivalent functionality)
* Explosion and bullet management
* Game play is independent of platform capabilities (i.e. uses a model tick approach)
* Game cycling e.g. detection of win / lose conditions and restarting the game
* There is a scoring system with the score shown on screen

**AI**

* Several different enemies with differing behaviours

**Report**

* This report has insightful and balanced reflection

**Extra Marks (Examples)**

* Mapping of world space on to screen space
* Other graphics techniques have been implemented e.g. background scrolling, blending modes etc.
* Level data is loaded from a file
* A difficulty level
* More advanced C++ e.g. use of namespaces, STL, C++ 11 and further patterns
* A\* algorithm
* ‘Intelligent’ enemy behaviour
* There are sound effects for collisions, explosions and firing
* Additional black box systems have been implemented e.g. for AI, Sound
* HUD features beyond simple text e.g. health bars, mini maps etc.
* Other features, please list below:

Maintenance Guide

**Visualisation System**

This system takes care of sprite and map creation and rendering. The class contains functions for initialising a game screen, grayscaling the screen, colouring the screen, creating sprites and rendering sprites. This class is used in the world system (next system that will be talked about) to render the map, player and enemies easily. It would be easy to use in other games as it allows you to call some simple functions to place sprites on the game screen. This system is good because it’s simple, and does a good job at rendering sprites into a simple 2D game, however, its simplicity means it is limited in what it can do. Other functions could be added to it to allow different visualisation effects, but as of now it simply renders and creates sprites.

There are only two classes in this system: Sprite and Visualisation. The visualisation class contains all of the previously mentioned functions, and the sprite class contains a texture pointer, and some rectangles, for the sprite and for each frame within the sprite. It also contains two functions which load and render the sprites. The sprite class is linked to a rectangle class which allows the creation of rectangles with either the left, right, top and bottom coordinates or a width and height value. This class also contains functions for clipping rectangles, checking if one rectangle is contained within another or if it’s completely outside of another (for collision purposes) and contains a function that changes the rectangle position.

**World System**

The world system is the main system within this project, and is linked to every class. It uses the visualisation system to render sprites into the game, uses the entity system and classes to create different entities, such as the background, player, and enemies. It also uses a spawn system that spawns chickens in random areas of the game, within the game window. This system manages the sound effects in the game, and uses the rectangle class through the entity classes to check for collisions between the player and the chickens, and updates a score on-screen when collisions are detected. This system runs the visualisation system and passes a width and height parameter to it upon being run, to initialise a game window.

As mentioned earlier, this system interacts with all classes. The entity classes are used in this system to create different entities in the game, which then are used alongside the visualisation system to render and create sprites for each entity. This system is very good as it keeps most of the actual game code within it, and because it is in the centre of the project, any class can be accessed from it. This is also bad as to use this class you would need all of the other classes in the project, which are all required in order for the game to actually work.

Conclusions

This project has been a huge learning curve for me, and I’ve learnt a lot about c++ while working on it. I’m happy that I was able to create a semi-working game, which contains simple, yet fun animation, as well as fairly accurate collisions. I’d like to have added an end-game state to the game, where harmful enemies can occur, and if they collide with the player, they reduce the player’s health. This would not only add an endgame state to the game, but would also add to the gameplay aspect, making the game more enjoyable. I’d also like to have implemented more sound effects, such as multiple sounds for the player and chicken collisions, which would be randomly determined upon each collision, and would have liked to have gotten the background music for the game to work. I think in some areas the code is a little messy, and could do with being simplified, but overall I’m happy to have implemented most of the main things I was being assessed on to this game.

\*This project working in OL8 on machine 46975

References

The game sprites (player, chickens and background) were created by me in Adobe Photoshop.

The chicken sound effect was downloaded from freesound.org and was uploaded by Rudmer\_Rotteveeland and is licenced under the Creative Commons 0, meaning it is allowed for use by the public domain.

The background music was downloaded from freesound.org and was uploaded by vikuserro and is licenced under the Creative Commons 0, meaning it is allowed for use in the public domain.