Games Engine Construction Report

GAV2016

DEVINE, PATRICK (Student)

Teesside university  A0101165

Contents

[User Guide 2](#_Toc125043557)

[**Implementation check list** 4](#_Toc125043558)

[Maintenance Guide 5](#_Toc125043559)

[Conclusions 7](#_Toc125043560)

# User Guide

How to play:

Click on folder containing the application, go to the demo folder, click on the executable and the game will run. (You may need to click the make distributable executable first if there is no openAI.dll file)

Control guide:



W – Move up

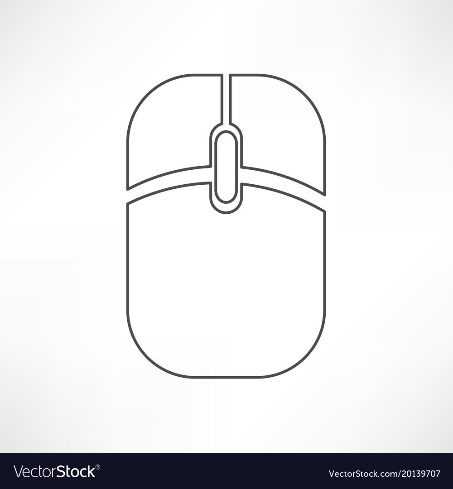
A – Move left

S – Move down

D – Move right

P – Pause game

Space – Dodge in the direction of movement



Left Analogue stick – Move

E – Stop fire

R – Start fire

Left click – Attack in the direction of movement

A picture containing diagram

Description automatically generated

Known issues:

Sound is manually called at the update using a Boolean variable set when collisions are detected, adding new sounds would result in too many variables being made in the header of the player entity.

There is currently no way to quit the game short of either getting the player killed or just exiting out manually.

### **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 configurations.
* 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 // **detects win and lose but doesn’t restart 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
* Difficulty levels.
* More advanced C++ e.g., use of namespaces, STL, C++ 11 and patterns
* ‘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:

The purpose of the visualisation system is to create and draw sprites, it can also draw rectangles given an area and handles collision checking. The collision checking is separate from the world system and therefore doesn’t stop the system from being black boxed. Since the visualisation is black boxed there is virtually no link between it and the game demo, therefore it would be easy to re-use in another project. There is some linkage to the HAPI system though (screen pointer is used to blit) so it would not be useable for anything not using HAPI as its base. The advantages of the system not being linked to the demo are: If any bugs occur with the graphics, it is a lot easier to fix as you can be sure it is a bug withing that system; the visualisation class itself can be used with any graphics system, meaning it would be easier to create a graphics system with a new API; it is much easier to create a new sprite using this system, all you have to do is call create sprite once and the system does the rest (no need to go through the world and find a value). The disadvantages of the system not being linked to the demo are: It is difficult to get information that is required from the visualisation system such as sprite width and height, which is why I decided to make the visualisation system handle collisions; not all sprites are the same, some of them are UI sprites and are not meant to interact with the world, if the visualisation system was linked to the demo it would be easier to implement this (by using a member variable like isInteractable in the sprite class); Harder to pass values into the functions in the sprite when having to go through another class.

Doesn’t store information other than what’s required to draw to the screen (like a screen pointer). All functions called in the visualisation find a sprite in its sprite map and call a sprite function.

Visualisation

1

\*

A class with four variables that when combined make a box, this box can be used to clip or detect collisions, every sprite has one box as their width and height do not change

BoundingBox

Sprite

1

1

Holds information about the sprite and allows the sprite to be drawn to the screen, can also check for collisions using bounding boxes

World Model:

The world model holds all the information about the game that is required to run it. The model also holds the game loop and handles detecting loss and win conditions. The purpose of this system is to check all the entities for input, update all the entities and render the entities during the game loop. The world model also handles loading of levels which uses a file to do so. The world model is linked to the game demo somewhat, while there are functions that could be re-used, a lot of the world model would have to be tweaked as not all games have the same win/lose conditions and members pertaining to the demo would have to be replaced with information stored in the entities. The advantages of the system being linked to the demo are: information is readily available to the demo; easy to make new entities and use them; passing values in are easy; world can be passed into the update allowing for easier communication between classes. The disadvantages of the system being linked to the demo are: if bugs occur it is harder to debug; making a new world model would require more work; not as re-usable.

Contains game loop and all objects contained within the game

Contains information and functions on in-game sprites and UI

World

Entity

\*

1

Inheritance

1

Handles sound (black boxed)

1

1

1

Sound

Visualisation

Input

Update and check collision are pure virtual, so they are different for each child

Handles graphics

Handles all input (returns a Boolean or int value based on input)

Background Entity

Enemy Entity

Player Entity

Bullet Entity

UI Entity

Explosion Entity

# Conclusions

The final implementation while functional is not what I had envisioned. I had planned to have multiple levels and another type of attack and a boss fight. However, due to time restraints and poor organisation I was unable to complete this work. There are also multiple features which (while functional) could be improved such as loading a level from a file; first of all, the code is overly complicated and could be made to be more readable and also the loader only loads entities (not all information about entities is loaded as well like state so saving and loading is impossible). If I were to re do this module the main lesson I’ve learned is that classes need to be organised more and work you do has to be written down and organised. This means that if I make a function that I plan to improve in the future I will not forget to do so. Forgetting about functions I wanted to improve, and just overall lack of direction is the main reason I was not able to complete the demo to the standard I wanted. In the future I would like to improve level loading to allow for saving and loading, add functionality so that the player can traverse different levels with the last level having a boss fight. The boss would have a move set instead of one attack so AI would also have to be improved.