Maintenance Guide – Ryan Jones P4019410

**The systems**

**Graphics**

The Graphics system is used to load in and render graphics of all kinds, as well as creating and rendering sprites in certain ways.

The Graphics system is black boxed from the other systems that it can be used in other games easily. The World Model calls functions from the Graphics system, but the Graphics system does not need to call any functions from the World Model. It is only dependant on the support classes Rectangle and Vector2 to fully function.

This system being black boxed away from the rest of the game means that it can be easily taken and used in other games without having too many changes. It also means that if it were taken and put into another game, it would need to be a 2D sprite based game. There is no functionality for other graphics.

The fact that the graphics system is only required to be put into the World model and Entity base class within this project means that there are very few errors (if any at all) from the Graphics system ending up outside of the Graphics system itself. The error handling within the Graphics system prevents any of this. The World Model only calls functions from the Graphics system, and never the other way around. The issue with this, however, is that even though the Graphics system is unlikely to fail, it does mean that there is a chance that if used incorrectly could prevent the rendering of everything.

Within this system, there are 2 graphics classes, which are assisted with the two support classes:

**Graphic classes:**

* Visualisation – This class is used to render everything in the application window. All the functions within it are used to either render a texture on to the screen at a given point or the creation and rendering of sprites. This class has direct access to the screen pointer of the application window, so in order to compensate if the pointer ends up outside of the screens memory the render functions are always clamped back on to the screen (although positions and textures can still be moved off screen, they just won’t be rendered). These sprites are contained within an unordered map in the Visualisation class.
* Sprite – This class is used to store a sprite loaded in to the Graphics system, and is able to render the same sprite multiple times. There is functionality built within the Sprite and Visualisation classes that allows the use of animated sprites by loading in a sprite sheet (in the case of this system, a vertical sheet) and splitting it by the given number of frames. A frame rectangle moved to show what part of the texture would be rendered at that point.

**Support classes:**

* Rectangle – This class is a support class used for collision checking and clipping. It is essential for the Graphics system as it is used to clamp the given rectangle and clip the texture so that it is still partially rendered whilst it is partially off screen. This class can also check if the rectangles of the sprite/texture are completely inside or outside a given rectangle (usually the screen) and is used within the Graphics system to either render the sprite faster or not at all depending on the sprites position. It is also essential to the World Model class for the functionality of collision with the same idea for rendering on/off screen.
* Vector2 – This class is a support class mainly used when the World Model requests a sprite to be rendered at a given position, which is given as a Vector2. To support that functionality, Vector2 would also be used within the Graphics system to mirror the same position, although at the time of writing this the Graphics system has not been updated with Vector2, so it is all converted within the World Model from Vector2 to using Vector2.x and Vector2.y when calling Graphics system functions.

**World Model**

The World Model system is used to load in and manipulate the world given to the player, and allows the player to manipulate the world within their limits. No graphics are attached to the World Model itself, and so calls the Graphics system whenever something is required to be rendered.

The World Model system is linked quite heavily to the game demo and so would therefore not be easily reusable in other game systems. It is also dependant on the Rectangle and Vector2 support classes, although they are independent and can be easily moved with it.

With the linkage being mostly dependant on the Graphics system, is does mean if a new Graphics system were used, so long as it is another 2D sprite-based system, it can be used for a multitude of different games. Being as it is dependent on the Graphics system, it also means it is restricted on how it is used with other Graphics systems. This World Model system being used for a 2D shooter, the only way it could be reusable is for another 2D sprite-based game, limiting the usage of the World Model.

By it being mostly dependant on the Graphics system, it does mean that with use of games such as the one used as the demo, it means that it is tailored to this type of game – a 2D scrolling shooter. The usage of sprites and rectangles means collision between entities is easily detectable and could be made better to be nearly pixel perfect. The issue with this, however, is that it requires the Graphics system to be kept nearly the same, save for a few small changes available occasionally. This is due to the collision detection and entity sprites are dependent on the frame rectangle of the sprite. Altering the Graphics system to use something other than rectangles would then require the World Model to be changed to use whichever new system is used. Additionally, all of the World Model positions are set assuming the screen size is 1024x1024, and would only work for that screen size.

Within this system, there is the main World class, the Entity class and seven child classes which inherit from the Entity class:

**World Model:**

* World – This is the main class within the World Model. This class is only called from within the World Model and the main demo class (which just creates an instance of the World and uses the World’s run function). This class manages a multitude of different aspects of the game, such as: Game tick functionality, score, game states, music and all entities. This class also updates the World and all entities within it after every game tick, which for this demo is set at 50 milliseconds, or 0.05 seconds.
* Entity – This is the base class for all entities within the World. This contains many variables which are used by all the children which affect and are affected by the game updates and other entities. There is also a “Faction” functionality which is set when an entity is created within the World. This faction is used to check which side all entities are on and subsequently check if they can collide, and if they do then both entities damage each other with their “damage inflicted” variable. This base class also uses a render function which calls from the Graphics system to render the sprite given to the entity. Additionally it has a respawn function used by the World class to respawn both enemies and players to their initial position.
* Background – This is an entity used for the background of the game. It has a unique faction which never collides with anything, and subsequently is never affected by any other entities. The only update this class does is that it moves across the screen at a rate given.
* Boss – This is the entity used for the unique boss enemy at the end of the level. It has some basic AI used to move from its initial position to a certain position on screen and just moves up and down. It also fires bullets every certain interval (which is faster than the regular enemies).
* Bullet – This is the entity used for all the bullets within the game. They are set as neutral and dead when starting within the game but as they are fired their side, initial position, initial direction and damage inflicted are determined by the entity that fired it. As it updates it just moves in the direction fired. After it is destroyed, it is set as dead again and put back into the pool of usable bullets.
* Enemy – This is the entity used for the regular enemies that appear within the world. Each enemy has a random speed moving up and down but they all move at the same speed across the screen. These enemies also fire periodically, although at a slower rate.
* Explosion – This is the entity used for all explosions within the game. They are set as background entities as they’re not intended to be used to collide with other entities, as well as initially dead as there’s no need for them at the start of the game. They only appear when an entity dies, and only appear for a small lifetime before they are set to dead again. They have a similar structure to the Bullet class, in the sense that they both use a pool of dead entities and are only set back to alive when required.
* Player – This is the entity used for the player to control. The player can control this entity with the keys given so they can move this entity on the screen, as well as shooting bullets from the entity. The player themselves cannot edit any data or member variables within the entity classes. This entity only has one health so when it collides with anything, it will be destroyed.
* Wall – This is the entity used for the environment within the world. The only functionality that is within this class is that it moves across the screen, and when it moves off screen, it appears back at the other side and continues to loop. They have a high amount of health so it is possible to destroy them as only the player and player bullets can collide with it.

**UML Diagram**

This diagram shows the relationship between the World Model and Graphics systems, showing that the World and Entity classes call the Visualisation class and is dependent on it for things such as rendering and collision detection. However, the Graphics system does not require the World Model system at all.

Within the World Model system there are many entities inherit from the Entity base class, but each of these could be used a number of times within the World. The same can be said for the sprites within the Graphics system, as the Visualisation has a collection of unique sprites that can be used a multiple number of times.

