Console Game Development Coursework 1

Introduction

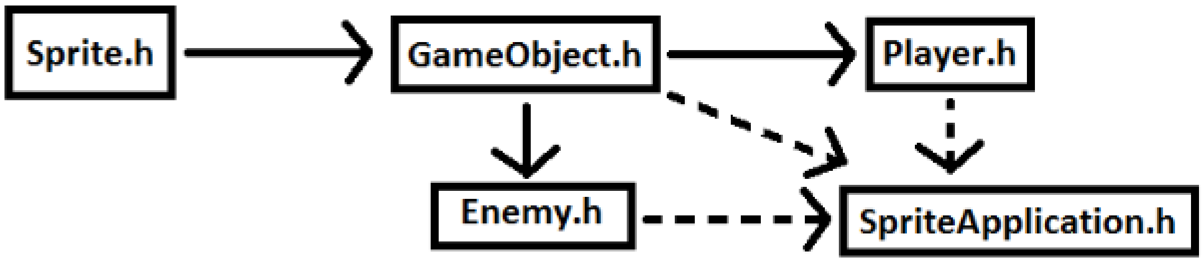
For semester 1, the task was to create a "2D sprite based console game for the PlayStation Vita" (University of Abertay Dundee, 2013). This suggests a simple game involving one or more sprites on screen which interact in some way, with user input controlling one or more of these sprites. The application that was created in response to this is a 2D side-scrolling beat 'em up. This includes a player-controlled sprite, an AI controlled series of enemies and a scrolling background. Although not in the coursework specification, the application includes textures for each sprite as a way of including mechanics which would otherwise be inaccessible. Both the player and enemy sprites have a feature set of 'attacks' which deal damage to the other. The application design and the techniques used will be discussed throughout this report. Techniques such as; collision detection, texture rendering and vectors. Alongside this, a user guide explaining how to run and use the program will be provided as well as critical analysis of the practices implemented.

Application Design

The first task was to design the application. Taking the coursework specification into consideration there were only a few examples of games which came to mind: Space Invaders, Pac-Man, Pong etc. Thus the conclusion was that textured sprites were to be used. This allowed for considerably more games to be considered for selection as inspiration for the application. In the end, side-scrolling beat 'em ups were the genre of choice, following iconic games such as: Streets of Rage, Dungeons and Dragons and Golden Axe amongst others. This would be approached in a rather simple manner.

The application consists of four classes: 'SpriteApplication', 'GameObject', 'Enemy' and 'Player'. The first two were given in the starter project 'cw1_sprite_app' and these have been developed over the weeks during labs, now consisting of collision detection, texture loading and several move functions. Alongside these, the Player and Enemy classes inherit from the GameObject class and expand on it by including an Enum state for the relative states needed. Not only that, but collision reactions and of course the enemy AI are central to these classes and keep the 'SpriteApplication' class clean.

As previously mentioned, inheritance has been utilised to fully maximise the feature set of the game. The main structure of the classes comes from the Abertay Framework (abfw). ‘GameObject’ inherits from ‘Sprite.h’ which allows for the creation and manipulation of sprites on the Vita. As it can be seen in the diagram below, both the ‘Player’ and ‘Enemy’ classes inherit from ‘GameObject’ which adds greater movement functionality and several Getters and Setters for ease of use whilst manipulating the sprites. This not only allows them to use functionality from the ‘GameObject’ class but also the ‘Sprite’ class. Finally, ‘SpriteApplication’ has access to all previously mentioned classes and functions involved therein through class definitions.



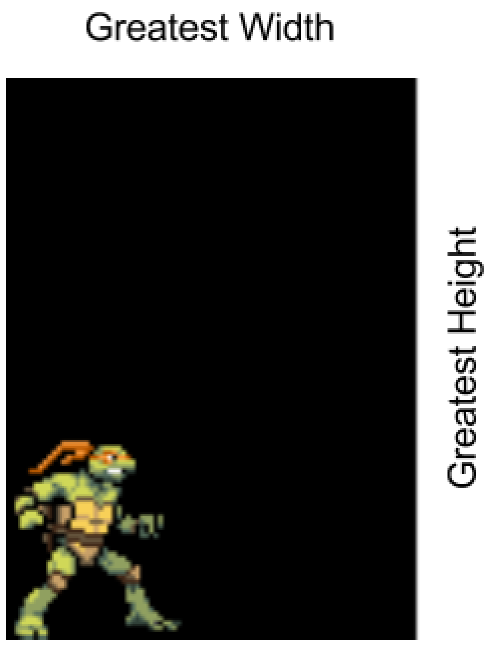
This has been implemented in such a way that the sprites can be altered with very little time and effort and keeps everything separate for each case. In ‘SpriteApplication’, this class structure also means that when developing new features for the other classes, they instantly have full access to the feature set.

Lastly, the choice of data structures was integral to the application as a whole by impacting the way the majority of the application was coded. Enumerated types were used to determine which state the player and enemies were in at any given time, whilst a vector was used to store multiple ‘Enemy’ objects which provided many benefits over alternatives such as arrays and multiple class definitions. Of course, the ‘Player’ and ‘Enemy’ classes themselves contain several values themselves relevant to each iteration of the class definition, meaning that the code continued to be organised and helped prevent repeated code.

<u>Techniques Used</u>

Several programming techniques have been used to meet and expand on the coursework specification. Collision detection, texture rendering and vectors amongst others are all techniques implemented and later discussed.

Collision detection for this application took some time to get working. This is due to the nature of the sprite textures used. Had all the sprites been the same size, collision detection could have been a simple case of bounding spheres/ boxes which contain the base sprite and use the base sprite variables. However, in order to maintain consistency on the spritesheet, the greatest width and height of the sprites were taken to give a standard size and all sprites were then contained within. This meant that the collision detection had to use custom values for the sprite width and height for each character state. In the end, bounding boxes were used to create a satisfying effect for collision but came at the sacrifice of previously given variables such as the sprite width and height getters. Following on from this, the collision reactions were simple to implement; player invincibility, player colour and sprite alpha values were altered to give a flickering effect. The flickering in particular hails back to the classics where to show the player that the character was hit it would flash for a few seconds with a small amount of invincibility.



As mentioned above, textured sprites have played a large role in making this application what it is. By influencing the original design, collision detection and gameplay etc. Textures are loaded on to the Vita through a relatively simple process. The image is loaded into 'image_data', a class which converts the .PNG information into code, and then is linked to the desired sprite. Considering this applications use of spritesheets, UV width and height have been implemented. To prevent confusion, when getting the required position on a spritesheet, the UV variables are used instead of the standard XY position variables. Three spritesheets were used in total; one for the player, enemy and background. These textures allowed for alternative gameplay mechanics to be included in the game, had the application followed the coursework specification directly and not have expanded on it then the game itself may have been very basic.

Furthermore, Enum states have been used to determine which animation state the characters are in. Although Enums and Switch cases are fundamentally the same, an Enum was simply chosen down to familiarity. An if-else ladder, although untidy, is easy to read and understand. Although a Switch case may allow for cleaner code, it would take a fair amount of disruption to the code in order to implement it. Thus, at least for the time being, an Enum will be the method of choice to control player and enemy states.

Lastly, since multiple enemies were needed, an array was considered to store the necessary information. However, a vector seemed more appropriate for this application due to its list capabilities and dynamic size allocation. Despite the fact that an array would have done the job, it would have meant extra variables in the Enemy class to determine whether or not a particular object was active or not. However, the list capabilities of the vector solves this issue by simply being able to reset Enemy objects at the desired time. Thus removing the need for unnecessary booleans and variables. However, this has caused a few problems. For example, the 'SpriteApplication' class is now full of for loops to cycle through all enemies. This is not only repeating code but time consuming and potentially detrimental to performance. However, arrays would have faced the same problem.

<u>User Guide</u>

In order to run the application, the main importance aside from the obvious development kit and 'abfw' necessity, is to have the correct files in the 'bin' folder. Due to the use of textures, these must be included in the folder otherwise the program will not run. The necessary extra files include: 'tmnt.png', 'Shredder.png' and 'Batman.png'; past the given 'bin' folder from the starter project.

When the application is running, the controls are straightforward. Use the directional buttons to translate the character left and right. The Square button will execute a basic attack, dealing a small amount of damage to the enemy. The Triangle button also deals a damaging blow to the enemy, this time with a little more force. Again, the Circle button deals damage but this time in the form of a jumping attack which will launch the character into the air and forward, dealing significant damage to all enemies in the players wake. Finally, the Cross button will allow the player to block incoming attacks from the enemy; although the player will always be invincible whilst certain frames of the attack animations are playing.

The application will end if the player dies.

Critical Analysis

The coursework specification states that the application must feature certain elements in order to comply with a standard. The first and foremost requirement is for the application to use coloured 2D sprites. This is done multiple times over through both the Player and Enemy objects as well as the background being exactly that. This is taken a step further by, as previously mentioned, the inclusion of textures. Furthermore, it is also required that the PS Vita controller is utilised in some form. This is done, as described in the user guide, through the simple use of buttons to translate and transform the player character. Otherwise, the program fulfills the specification by being a playable game with sprites and player controls at its base level.

Throughout the project a few limitations and several highlights were faced. One limitation which severely lengthened development time was the fact that currently there is no PC compiler for the application. This meant that in order to continue work properly and be able to test the application; the labs were a complete necessity but due to other classes requiring the use of the computers, this was not always possible. Aside from this, another limitation was the frameworks' coding style (although of Google standard) was significantly different and took time to get used to. However, the highlights of this task by far outweigh the limitations.

Gaining experience using a framework is an enormous benefit. It not only gave more than enough functionality to the Vita to be able to complete the application without many problems but also served as an encouragement to push the application further. Moreover, the coursework specification itself left much to be desired and therefore encouraged to push forward and create something that surpassed the minimum.

Due to the time constraint on the coursework, there a few areas where the code could be improved upon. For example, it is likely that the Player and Enemy classes could be combined into a single class which takes alternative types of input for the necessary situation i.e. AI for enemy objects and player input for the player object. Secondly, the spritesheets used for the 'Player' and 'Enemy' textures could be combined into a single, compressed .PNG to save time and memory when loading them in. Moreover, the spritesheet must comply with certain guidelines, namely conforming to a power of 2 width despite there being nothing (at least nothing that is obvious) clearly stating this in the official Sony documentation. This now means that the sprites, at least in the player sprites case, the animation stream no longer fills the entire width of the file. Thus causing problems in the animation function where it uses exact values to calculate how many tiles to cover. Furthermore, the ‘SpriteApplication’ could be cleaned up to make the code slightly easier to read and understand, although it has been cleaned up to a certain extent, due to previously mentioned improvements this is limited.

Lastly, there were several alternative strategies that could have been employed. A class-based structure was chosen against, for example, simply having everything in functions in ‘SpriteApplication’ not only to prevent repeated code but also to make the code much easier to read and understand. Classes cuts down on the length of the code and allows for multiple objects on screen to be the same and independent through the use of arrays, vectors and multiple class definitions. Although, as previously mentioned, the ‘Player’ and ‘Enemy’ classes could have been combined, but were kept separate for the purposes of this coursework to ensure a solid, working finished product.

Conclusions

In conclusion, the application meets the coursework specification by including 2D sprites and player controls and surpasses it with the inclusion of textures. It saw many iterations to see the final product complete which struck several hurdles and sighs of relief. Doing this task again, more time would have been spent on planning the application and working out what exactly was needed, as opposed to coding almost blind from the start. Had this been done then the problems faced with collision detection, lab time and everything else may have been avoided. Despite these shortcomings, working with the Vita and using the framework gave great experience and are great platforms to work with. Certainly any future coursework will involve considerably more features and again experience with the Vita.

References

University of Abertay Dundee, 2013. *AG0801ACW1.* Dundee: IAMG.