



C03015 Computer Science Project

Android Game Using an Entity Component System

Dissertation



April 15, 2017

Asamari Egwu

Department of Informatics, University of Leicester

DECLARATION

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s).

Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s).

I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Asamari Egwu

Signed: 

Date: 7/12/2017

Contents

Abstract.....	1
Introduction	2
Requirements.....	3
Functional Requirements.....	3
Non-Functional Requirements	4
A Brief Showcase of The Project	5
Overview of Software, Architecture and Design	9
Entity Component Systems (ECS).....	9
What is a Component?.....	9
What is an Entity?	10
What is System?.....	10
So why use an Entity Component System?.....	11
Implementation of an Entity Component System.	12
LibGDX and Artemis ODB	15
LibGDX.....	15
Artemis-ODB	15
Scene2D UI	16
Observer Pattern.....	17
Pathfinding Algorithms	17
Utility Theory and Artificial Intelligence (AI).....	19
Map and Event Generation.....	22
3 rd Party Systems.....	25
Critical Appraisal	26
Summary and Analysis of Work Completed	26
Commercial and Economic Context.....	28
Personal Development During the Project	28
Conclusion.....	29
Bibliography	30

Abstract

In game development using traditional object-oriented design can make changes to code difficult as the size of the project increases. Refactoring child functionality into a parent class, may be a solution but the process can become difficult to maintain.

The game design process ever evolves, and it can be difficult to plan every single interaction and change you are going to have when you first start.

In this project I investigated using an Entity Component System (ECS) to create a skeleton of an android game. An Entity Component System is one where all game objects become 'entities' and are broken down into different components. For example, a bottle could be made out of a 'Position' and 'Container' Component. A book could be made out of a 'Position' and 'Readable' Component. As both objects can be placed, but they have different functionality.

Systems then interact with different entities based on their components. For example, a 'Movement' System may be set to only interact with objects with a 'Position' Component. Which means the bottle and book could be edited by the Movement System.

The essence of ECS is to find the commonality between game objects so changing an entity from one thing into another thing is a lot easier than with traditional object-oriented inheritance.

The game I finally, settled on making is a turn-based strategy game involving protecting your base and defeating enemies trying to attack it. You travel through different scenarios to eventually confront the final boss protecting the health of your player-controlled units. In which after defeating said boss, the game ends.

//Notes: Is it a good idea in the abstract to mention other technological aspects of my game? Such as the ones shown below? As they weren't part of the mission statement, but cropped up over time based mostly on my choice of game.

The game also utilizes other concepts such as a score-based decision AI, also known as 'Utility' AI. Pathfinding algorithms, such as A*. Also, the game is randomly generated upon starting so each run through should be slightly different.

Introduction

Game development is an area I've had a small interest in for quite some time. Due to this when researching how to create my own games I stumbled across a different way to create a game. It was called an Entity Component System (ECS). This architectural pattern is a different way of building and creating game objects. Thief: The Dark Project (1998) was one of the first games known publicly to use this architecture. Over the years it has increased in popularity, some game engines have been built using this idea as it's core.

The core principles of ECS will be explained further in the dissertation.

The aims of my project were:

- To create a 'skeleton' game which utilised ECS to build and create game objects and their interactions with the game world. The use of 'skeleton' I did not plan to create a fully fleshed out game as given the time, I knew from the start the use of core principles ECS is what I wanted this project to be about, not the actual game itself.
- Build a game, which uses intuitive controls for the platform I selected. (Mobile). A user should be able to interact with the game without too much confusion.
- Build the game using both LibGDX and Artemis-ODB frameworks. The reason for choosing these frameworks will be explained further

The objectives of my project were to:

- Build a core gameplay loop, which would be the frame I created game objects for. An example of a gameplay loop would be one in Chess, first it is a player's turn, they move a piece. Then it is the opponents turn, they move a piece. This continue until there is a victor. If the project didn't have a gameplay idea in place, it would be difficult to build anything for it.
- Finish with a working and functional product. This is an over-arching objective. I wanted the game to be played until a conclusion. To be in a state where it could be 'completed'. Games are notorious for over-scoping and being able to finish in time, was an objective I wanted to keep.
- The control scheme of the game need to be intuitive to use. This could be done using a tutorial or using commonly known visuals. The game needed to be able to 'speak' for itself and the player needed to be able to understand.

I faced several challenges when trying to adhere to both the aims and objectives of the project. The biggest I feel is that, in essence, this was a 'creative' project. The canvas may have been too blank, as although I wanted to focus on the principles of ECS. I still needed to also create a game of my own. Perhaps a better alternative, may have been to re-title the project. 'Tetris: Using ECS'. Create a known game, but use ECS architecture. As the technical aspects of the project would remain the same, and wouldn't be restricted by my own potentially lacking creativity. I'll discuss more in the reflective section of my dissertation.

Requirements

Below is the list of requirements I initially creating when starting my project. In the list the requirements that have changed over the course of the project have been marked with a (*) and their changes will be written below.

//Note: Here I reference, changes made over the course of the project at this point I haven't actually //shown the game. Would it be better to just detail all the requirements, and then later reflect what //has changed about them?

Functional Requirements

Basics

- The Player can select different modes of game to play*
 - The different modes of the game have been changed to instead pick different characters to play through the game with.
- The Player can view their statistics
- The Player can view their character*
 - Characters. A player controls multiple units instead of just one.
- The Player cannot save their game during combat
- The Player can pause the game
- The Player can exit the game

Player Character

- A player can control a single player character*
 - Multiple player characters.
- The player interacts with enemies by either tapping, swiping or dragging them.
- If the player's character dies the game is over*
 - An additional resource has been added,
- A character's animation responds based on player input (stretch)*
 - Not included

Gameplay (Needs to be tested)

- The game will have a player turn and enemy turn
- During the player turn the player can attack enemies
- During the enemy turn the player can defend themselves
- Enemies will be defeated when they run out of health
- Arenas can hold one or multiple waves of enemies
- A player can recover loot from enemies or chests*
 - Player rewards are recovered after a battle has taken place.
- A player can activate special moves that do more damage to enemies

- Players both attack and defend themselves using touch controls

Map

- The map will display different rooms to players
- The map will display special rooms in different colours to players
- The map will highlight the current room the player is in

Non-Functional Requirements

- The game utilizes both LibGDX and Artemis ODB.
- The game utilizes both sound and music
- The game utilizes a range of touch controls, (swiping, tapping, dragging)
- The game can be installed on an Android Phone
- The game can be run on minimum Android SDK version of 15 (changeable)
- The game can save and load user data
- Mobile processors should be able to render the game without any significant drops to frames
- The game controls should be intuitive to users, easy to pick up and use
- The game can be hosted on the Play Store without content issues
- The game will have item customization for characters

A Brief Showcase of The Project

Before moving on to describing the different software and architecture and researched and used to create my project. I first wanted to show and explain different parts of the front-facing side of the game. This is to help with visualization when describing the back-end.

A summary of the game loop I created is this:

- A player first decides which character they want to pick to complete the game.
- A player then must travel through a map, from the left of the screen to the right, until they reach the end.
- A player is faced with different battles that they need to complete. During these battles two resources are threatened which determine if a player loses the game or wins.
- These resources are the player's characters' health. And their morale. If either of these numbers reaches zero, the game is over. And the player needs to start from the beginning.

Here are some images of the game in action.

This is the Menu Screen before you enter the game proper. You select which characters you want to be in your party You can only select one from each group of characters.



Figure 1: Character Selection Screen

This is the game map. It is randomly generated for each game. In order to traverse to the end of the game you need to tap on one of the bright white nodes and that will trigger an event based on the image. The ones highlighted below will trigger a 'Battle' event where player will face off against enemies.

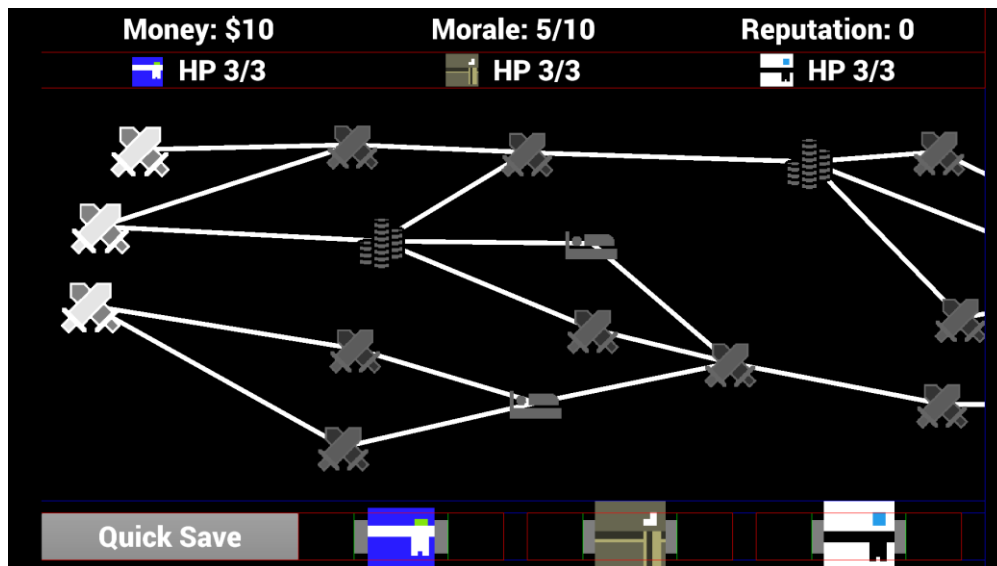


Figure 2: Map Screen

This is a battle. When a battle first begins a players needs to select where to deploy their units, within the blue zones. A battle has objectives (shown on the right) and within a battle you need to protect your bases, while defeating waves of enemies.

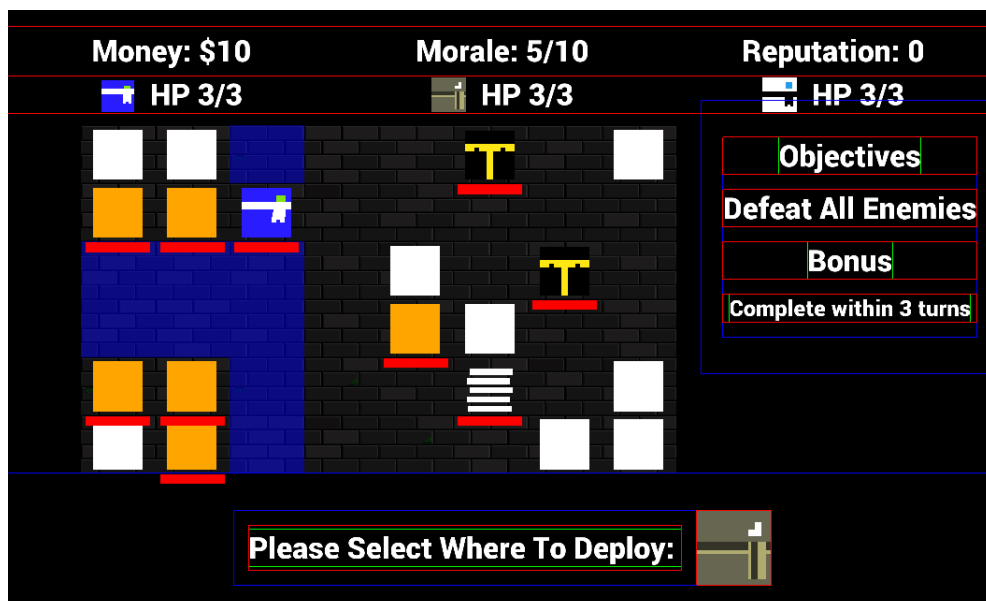


Figure 3: Battle Screen (Deployment)

You can select your characters and their skills. Enemies, move and telegraph where they will attack when a turn is over. It is up to you to push/stun/defeat your enemies so they do not attack either you, or your bases.

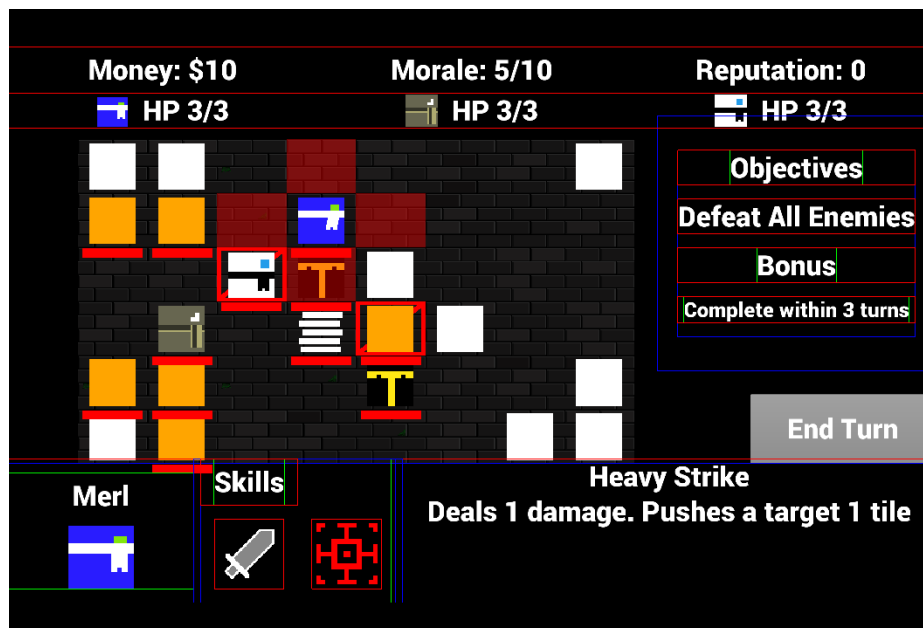


Figure 4: Battle Screen (Player is attacking)

This is what a shop looks like within the game, you can buy and sell items.



Figure 5: Shop selling various skills

Here is a rest site, where a player can decide to either heal their characters or heal their 'Morale'.




Money: \$10	Morale: 5/10	Reputation: 0
 HP 3/3	 HP 3/3	 HP 3/3
You've Reached a Rest Site		
Please Select An Action to Take		
Rest (Restore 1 hp to all party members)		
Boost Morale (Restore 1 to your party's morale)		
Leave		

Figure 6: Rest site. Restore Health or Morale?

Overview of Software, Architecture and Design

Initially, this project was focussed solely on ECS architecture, however as the game developed and I started learning more towards a strategic video game. More was required to see the game to fruition. Design patterns such as Observer and Utility AI. As well as looking into pathing finding algorithms. For each major technical aspect of my project I will detail the research and implementation of it within the project. Starting with the most important, which is the ECS.

Entity Component Systems (ECS)

My original reason for doing this project is that I heard about ECS when talking with a friend who was developing their own game. They talked about how they had initially found a bit of trouble inheritance and that they'd started to investigate using ECS. I didn't know what that was, but was interested in making games at the time and asked him about. I then went on to do further research I figured it was what I wanted to base my project on.

To explain ECS properly you must first break it up into its 3 words. Entity, Component and System.

What is a Component?

The purpose of a component is to 'Hold pieces of game data, but not game logic through them'[6].

An example of a component can be seen below.

```
public class ExpireComponent extends Component {  
  
    public float expiryTime;  
  
    public ExpireComponent() { this( expiryTime: 0); }  
  
    public ExpireComponent(float expiryTime) { this.expiryTime = expiryTime; }  
  
}
```

Very short and very simple. This component only holds data and is unaware of anything to do with the game. Component can be edited by an external force but internally, the best practice is to ensure they are only aware of themselves.

What is an Entity?

'Entities are a collection of components' [6] The soul purpose of an entity is to act like a bag of sorts. It stores as many components as you require it to and those components define what it is in the game world. An entity also does not know what it is. It has no reference to that.

An example of this can be seen here. If you wanted to build a tree in your game it might be as simple as doing this:



What is System?

Systems typically operate on a group of entities that share a specific set of components. It is in systems where the game logic is created. [6]

Systems act like 'gatekeepers' to their own functionality. An example of a system can be shown below:

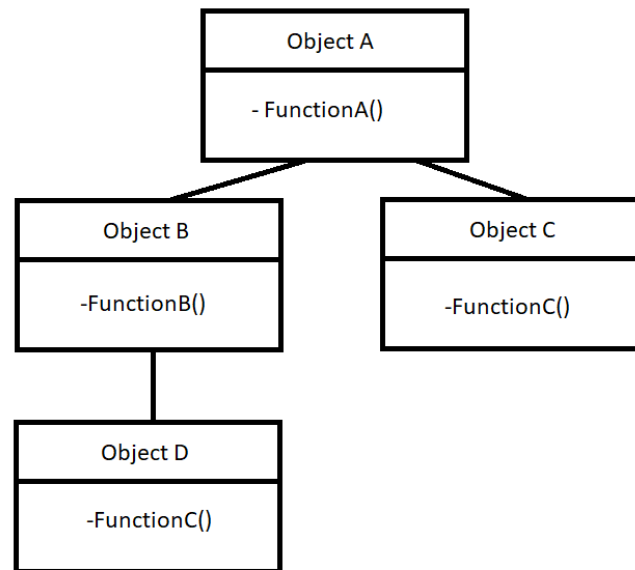
```
public class MovementSystem extends EntityProcessingSystem {  
  
    ComponentMapper<PositionComponent> pm;  
    ComponentMapper<VelocityComponent> vm;  
  
    /unchecked/  
    public MovementSystem() { super(Aspect.all(PositionComponent.class, VelocityComponent.class)); }  
  
    @Override  
    protected void process(Entity e) {  
        PositionComponent pc = pm.get(e);  
        VelocityComponent vc = vm.get(e);  
        pc.position.add( x: vc.velocity.x * world.delta, y: vc.velocity.y * world.delta, z: 0);  
    }  
}
```

The highlighted portion of the image shows the two components this system looks for in all entities that currently exist within the game world. Only entities that contain the Position Component and Velocity component are operated on by this system.

So why use an Entity Component System?

'ECS can circumvent the “impossible” problem of hard-coding all entity relationships at start of project’ [5]. A common problem amongst software is that it can be difficult to plan for unexpected changes in functionality. Object oriented programming tactics such as using inheritance can already help circumvent this problem.

For example, here we have a small problem:



One the key software principles in the D.R.Y principle. Don't Repeat Yourself. This tree contradicts this as both Object C and D contain FunctionC. Object D requires functionality from B and A and C, so what can you do? In this case a simple refactor is in order, to maybe push FunctionC up in the inheritance tree. There are many solutions to this. However, this is a simple tree. What if the tree had quite a few more levels than just 3?

ECS changes this problem into this:

Entity A – Component A,

Entity B – Component A, Component B,

Entity C – Component A, Component C,

Entity D – Component A, Component B, Component C

The functions are then handled by the systems instead. This increase the flexibility of your design, Entity D can turn into and function like Entity C, simply by removing a component. This can 'Allow a single entity to span multiple domains without coupling the domains to each other.' [11]

ECS changes the way you view objects within your game. You need to be able to break them down into similar components. One of the benefits of this system, is that new functionality can be created simply by making a new component and making a new system that only looks for that component.

In my implementation I called components like that 'identifiers' as they held no data, but simple by added them to an entity it would act differently in the game world.

```
public class EnemyComponent extends Component{}
```

This class would be an example of that. Simply by adding this into an entity it is treated differently by the game.

Now, there are downsides to an ECS. As with most software architecture and designs it is only as good as the person who built it. You can end up flooding your game with components and systems that aren't necessary or create components that aren't decoupled from other components. Which means it can become harder to maintain.

Implementation of an Entity Component System.

In an actual implementation of an ECS your entities are usually composed of a lot more components. Especially ones used the main parts of your game, such as player characters.

The Artemis-ODB framework comes with pre-set classes known as World, System, Entity and Component.

In short, Worlds process systems and Systems process entities that have components that the System is looking for. Here is an example of an Entity being built within my project.

```
public ComponentBag baseUnitBag(UnitData unitData) {  
  
    ComponentBag bag = new ComponentBag();  
    bag.add(new PositionComponent());  
    bag.add(new UnitComponent(unitData));  
    bag.add(new SolidComponent());  
  
    bag.add(new HealthComponent(unitData.statComponent.health, unitData.statComponent.maxHealth));  
    bag.add(new CoordinateComponent());  
    bag.add(new MoveToComponent(Measure.units(160f)));  
    bag.add(new VelocityComponent());  
    bag.add(new TargetComponent());  
    bag.add(new BuffComponent());  
  
    //Graphical  
    bag.add(new BlinkOnHitComponent());  
    bag.add(unitData.statComponent);  
    bag.add(new TurnComponent());  
  
    return bag;  
}
```

Now each of these components have different systems or set of systems that may interact with them. But let's focus on the system I showed earlier. Which interacted on the Position and Velocity Components.

```

public class MovementSystem extends EntityProcessingSystem {

    ComponentMapper<PositionComponent> pm;
    ComponentMapper<VelocityComponent> vm;

    /unchecked/
    public MovementSystem() { super(Aspect.all(PositionComponent.class, VelocityComponent.class)); }

    @Override
    protected void process(Entity e) {

        PositionComponent pc = pm.get(e);
        VelocityComponent vc = vm.get(e);
        pc.position.add( x: vc.velocity.x * world.delta, y: vc.velocity.y * world.delta, z: 0);

    }

}

```

In the Movement System it uses the velocity set on the current Entity to change its position. And this is all the system does. Of all the systems I've created I would say this is one of the smallest ones and most 'pure' as it only relies on itself and does a very simple task.

In Artemis ODB Systems are stored within 'Worlds' which then process them as can be seen below.

```

WorldConfiguration config = new WorldConfigurationBuilder()
    .with(WorldConfigurationBuilder.Priority.HIGHEST,

        //Initialize Tiles
        new TileSystem(battleEvent),
        new BattleDeploymentSystem(game, battleEvent),

        new BattleWorldInputHandlerSystem(gameport),
        new BattleScreenUISystem(UIStage, game),

        new PlayerPartyManagementSystem(partyDetails),
        new InformationBannerSystem(game, gameport),

        new MovementSystem(),
        new FollowPositionSystem(),
        new UpdatePositionSystem(),

        new BuffSystem(),

        new MoveToTargetSystem()
    )
    .with(WorldConfigurationBuilder.Priority.HIGH,
        new ConditionalActionSystem(),
        new ExplosionSystem(),
        new TurnSystem(),
        new HealthSystem(),
        new ParentChildSystem(),
        new BlinkOnHitSystem(),
        new ExpireSystem(),
        new PlayerControlledSystem(game),
        new EndBattleSystem(game, battleEvent, partyDetails)
    )
    .with(WorldConfigurationBuilder.Priority.LOWEST,
        new ActionOnTapSystem(),
        new ActionCameraSystem(),

        //Rendering Effects
        new FadeSystem(),
    )

```

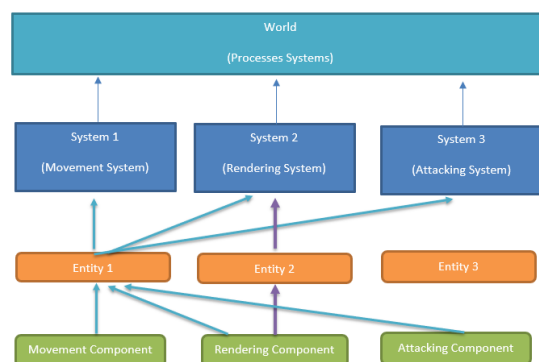
This is part of the BattleScreen's World and it is quite large, as the battle screen is one of the more complex and contains the game's main gameplay. Each System is processed sequentially from top to bottom and each looks for entities that are inside of the world that has the correct components for them to interact with.

Now, I'm I happy with my implementation? In honesty, it's hard to say. I've created a number of small systems that do their jobs and are easier to understand. However, some systems are quite unwieldy in nature.

In ECS Systems can talk to one another. So, some systems, rely other systems to work. However, I worry that in my code that may not be clear. I'm worried it might be tricky to give my code over to someone else and then tell them to continue where I left off. Even though I have comments there are some systems that I feel could be built better, and if I do decide to continue this game to completion I would want to look over and improve upon parts of my implementation. To increase both it's maintainability and usability.

It was late into development that I discovered something that might alleviate this problem, known as the 'observer' pattern. Which I will be discussing later.

//Note: Should I also include the image I used last time as an abstract implementation? The one shown below.



//Note: Also, I tried to bake in my research about ECS into this as well, as is shown as an option in the guidelines.

LibGDX and Artemis ODB

//NOTE: I'm quite unsure about this section. It feels like I'm waffling.

There are two precursors I used when I selected the game engine/framework I would use to create my project: Can it use an Entity Component System? Can it create games for the Android platform?

LibGDX

The reason I picked this framework was because its documentation is quite broad and well written, it is open source, and LibGDX also has two ECS frameworks that can be attached to it.

Additionally, LibGDX is 'lightweight' in comparison to game engines, as it is a framework. In LibGDX you are given the tools to create your own game engine easily, but it is not an engine itself.

Such tools include [10]:

- Cross platform integration (Android included)
- Audio handling (both music and sound)
- File I/O (Saving data)
- Graphics. (The complexities of drawing something to the screen is reduced to a simple draw call).

LibGDX also provides the ability to create menus using UI classes, which is a function I discovered at a late date.

It also comes with custom classes such as `Array<T>` which apparently run faster than Java's standard `ArrayList` class. This is important as games process many frames per seconds, so anything to increase performance is a good idea.

I originally liked the idea of using LibGDX as it provided less hand-holding than other game engines like Unity. It isn't as popular as Unity. Which means the level of examples or help online is reduced overall. There are resources online, as well as tutorials, but not as many as other engines, sometimes I wondered if what I was doing was the correct way to use the framework.

Still, it was an interesting challenge especially as the framework I used to utilize an ECS was even more niche.

Artemis-ODB

ECS frameworks that can be built with LibGDX include the 'Ashley framework' and 'Artemis-ODB' framework.

Both are quite niche and resources online on how to properly utilize them are few and far between, more so on the side of Artemis. However, their wiki pages and documents are quite filled with information that made it easier to use.

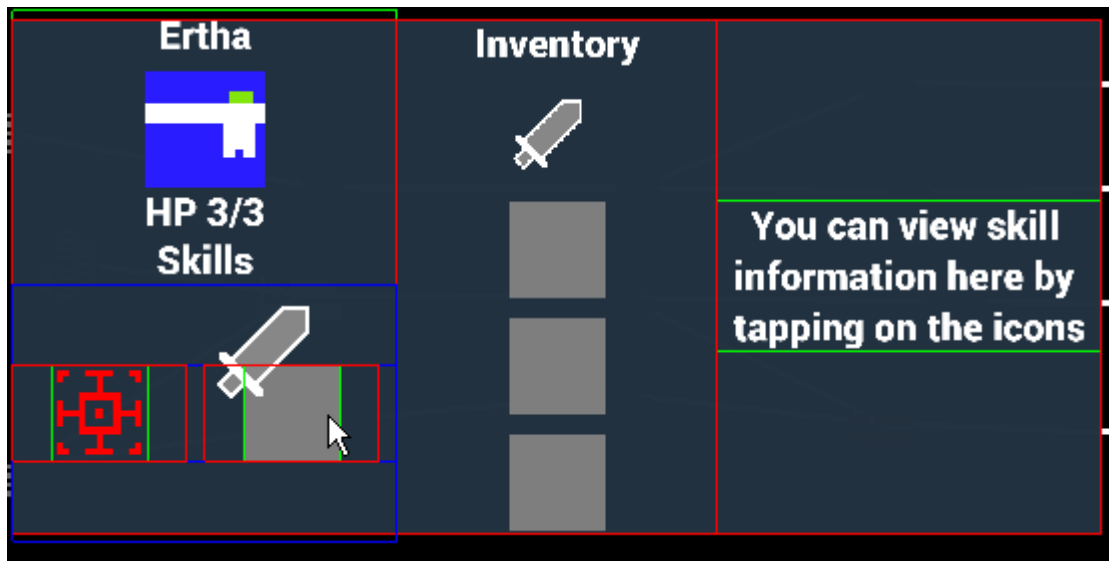
I ultimately decided to use Artemis due to the charts shown in [7]. However, during the project I looked at games that were using Artemis and I could certainly see the difference between a novice (myself) and someone who knew what they were doing.

Scene2D UI

This is a featured contained within LibGDX in which it uses it's 'Actor' classes but for the purposes of UI. During the project I had originally started hard-coding UI elements and slowly realized as the complexity increased, this was a bad way to do things.

I looked into how others created UI elements within LibGDX and came across this.

Scene2D UI is a package that enables you to more easily create Tables and different interactable UI elements. Tables are the primary way of doing formatting.



Finding this helped to alleviate a growing problem I found with my project as I took it down a different path.

The type of game I was creating needed more a lot more menus. In fact menus became quite important. As there was a lot of information that needed to be relayed to the player. Such as their skill information, party information and buying and selling items.

Observer Pattern

//Note: Was unsure if I should also mention this as well.

//I would basically talk about what is featured here:

<http://gameprogrammingpatterns.com/observer.html>

Pathfinding Algorithms

When the game made a shift to turn-based actions. I needed a way to calculate the paths player and enemy units may take to navigate through the battle map. Coordinates became the new positions. This meant I needed to research how to find paths. Mainly, the shortest path. This would also need to work through and around obstacles.

As I already knew about Dijkstra's I started looking for ways to implement it, when I came across the A* pathing algorithm. In a comparison between A*, Dijkstra's and Greedy Best-First Search algorithms [4], it was found when obstacles were introduced both A* and Dijkstra's performed better. However, A* was faster as Dijkstra's had to look at more co-ordinates.

To briefly describe how A* pathing works. It uses both aspect of Dijkstra's and a Greedy Best-First Search algorithm to find the shortest path. First it creates a 'h' or heuristic value for each co-ordinate which is determined by how far each co-ordinate is from the end goal.

It then calculates a g value which is determined by how far it would take to move from the start co-ordinate to the given co-ordinate.

From this value an f value is determined which is used to show where the next step should be. My implementation of a node is shown below:

```
private class Node {  
  
    public Node(Coordinates coordinates) { this.coordinates = coordinates; }  
  
    public Node parent;  
  
    public Coordinates coordinates;  
  
    public int hValue;  
    public int gValue;  
    public int fValue;  
  
    public void calcF() { fValue = gValue + hValue; }  
  
    private void setHeuristic(Coordinates goal) {  
  
        int distX = Math.abs(coordinates.getX() - goal.getX());  
        int distY = Math.abs(coordinates.getY() - goal.getY());  
  
        int D = 1;  
  
        this.hValue = D * (distX + distY);  
  
    }  
}
```

The D value is usually determined by 'the lowest cost between adjacent squares'[4]. Which is my case was 1.

As my project continued it turned out that not only did I need to know the shortest path, it terms of AI it was required to know all paths. All the time, this is because the game became more dynamic. There were moving parts. As I introduced more characters and structures for the AI to decide between it needed to know the best paths to almost every co-ordinate on the map. Every turn.

This change in scope, meant I needed to revisit and see if maybe Dijkstra's might be a better fit. However, upon further research it was found 'All game developers understand that A* is the pathfinding search algorithm of choice, but... it is not a panacea'. [13] What this means is that your implementation of A* affects how it performs. It is more difficult than Dijkstra's to implement but it is almost always faster, if you use a correct heuristic. As A* has been described as Dijkstra's but with a heuristic to speed it up.

Even though I use more pathfinding than I was nearer the start of the project the performance of the game is still constant. I can't say that my implementation of A* is better than if I was using a different algorithm but for it's purposes I believe the difference would be negligible. As, the game map itself is quite small.

As you tend towards much greater values of n this is where your choice of pathing would matter. For my uses, it isn't as important.

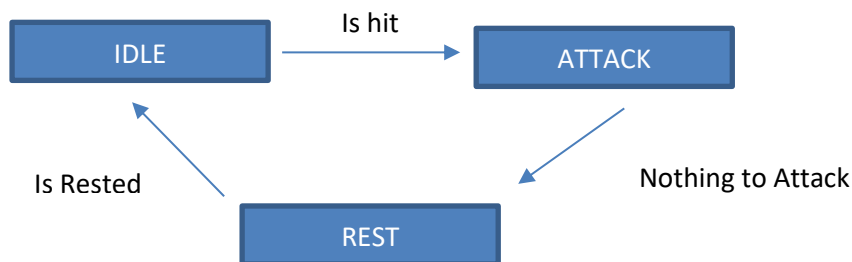
//Note: Reading the guide, it mentions I need to justify why I picked certain things, which I tried to //do here. But, do I also need to describe what A* pathing is?

Utility Theory and Artificial Intelligence (AI)

As the game tended towards strategy I needed a way to have the enemy 'think'. How would they decide who to attack?

I looked into different ways to do AI and a common approach was finite state machines and behaviour trees.

Finite state machines are simply having a set amount of states an enemy can be in. For older games with less advanced Ai, this was a common approach.



As you might imagine, states in modern games could balloon to hundreds of different states. It wasn't quite maintainable. In my project, originally a state based system may have worked, but I knew early on I wanted to try and have more complex and less predictable AI.

Utility Theory can be described as this, 'every possible action or state within a given model can be described with a single uniform value'. This value is the usefulness of the action. Or in the case of my project the actions most likely to be taken by an enemy, when it is their turn.

Below is an example, of an early attempt at Utility Theory on AI within my project.

The enemy had a total of three decisions and each decision had a set of calculations that determined the score for the decision.

Actions	Calculations
End Turn	Can I do anything? No = 100
Move Towards Target	Am I in Range to Attack? Yes = -1000 No = +100 Can I use my movement skill? Yes = +100 No = -1000
Attack Target	Am I in Range to Attack? Yes = +100 No = -1000

	Can I use my attacking skill? Yes = +100 No = -1000
--	-----------------------------------------------------------

In this instance if the enemy was in range to attack the scores would tally in the numbers shown in the table below

Actions	Total Score
End Turn	0
Move Towards Target	-900
Attack Target	200

This meant the enemy would attack.

However, I later found that this was too simple an approach. As there were other variables. Mainly, target became targets. Not only were there now multiple allied forces there were also allied structures that enemies needed to attack. Also, just being in range wasn't enough. You had to be in range and your attack need to be able to hit the target. One issue I had is that if I placed a wall in a certain way, enemies would refuse to move or attack. Even if there was a path available.

This is because they were in range to attack, so the movement score was lower. But they couldn't attack since there was a wall. All they could do was end their turns. Even though they should have chosen to move elsewhere instead.

I needed to overhaul my idea, when it came to using Utility Theory. I was making multiple classes, that were near enough doing similar things. But some were using ranged attacks, and others using melee attacks. It was difficult to maintain and I could definitely see there was a problem.

As Utility AI is a design-based AI, there is one major flaw: 'it will still rarely be smarter than the AI developer who designed it.' [1]

The changes I made involved pretending as if the enemies were players themselves. When a player decides to make a move, they look at everything. They look every enemy, every structure and every skill they have at their disposal. In their minds they weigh which co-ordinates are the best to move to. They decide this by figuring out which co-ordinates allow them to use their skill to attack which enemies they deem the most important at the time.

This is what Utility Theory is about. It is about pretending as if your AI is thinking like it is a human. Or in the case of my game. Think like it is a human playing my video game.

Every single turn before an enemy decides to move to calculates these decisions for every tile on the map.

- Can I reach this tile?
 - If I can reach it, can I reach this tile within my movement range?
- At this tile, if I used my skill which tiles would be affected?

- Of the affects tiles, which tiles contains a player character?
- Of the affected tiles, which tiles contain a player base?
- Of the affected tiles, if I were there I'm I being attacked by other enemies?

//NOTE: Should I included a visual representation of this?

All these decisions are calculated and the tiles with the best scores and paths and brought to the forefront. The enemy then moves to said tile and casts the appropriate skill.

Using this I was unable to trick the AI into being unable to move. The only time they did not move is if I purposefully created a map, which meant there was no path to anything that they could attack. This was a marked improvement over my first iteration.

However, this iteration is still flawed. It lacks randomness, or decisions that might not have been the best decision, but still a decision. By this I mean, enemies prioritize destroying player bases. But what if one enemy decided that they prefer to just attack players instead? A more advanced game may apply 'personalities' to their enemies to add flavour. For example, enemies who attack from range may try to avoid getting near player targets. They'd rather attack a base that wasn't close to any players.

I also, never got a chance to advance to more mathematical ways of generating Utility score. My scores are binary in nature. Where you could build a score that uses more advanced mathematics.

This is a good jumping off point however, and I feel that using Utility theory for AI is the way to go. It changes your thinking from states and if/else statements. To just thinking about your AI in a more human way.

You can ask yourself questions such as what would do in this situation? Based on the answer you can design scoring systems to reflect it. Designing for those choices makes it easier in my eyes.

Map and Event Generation

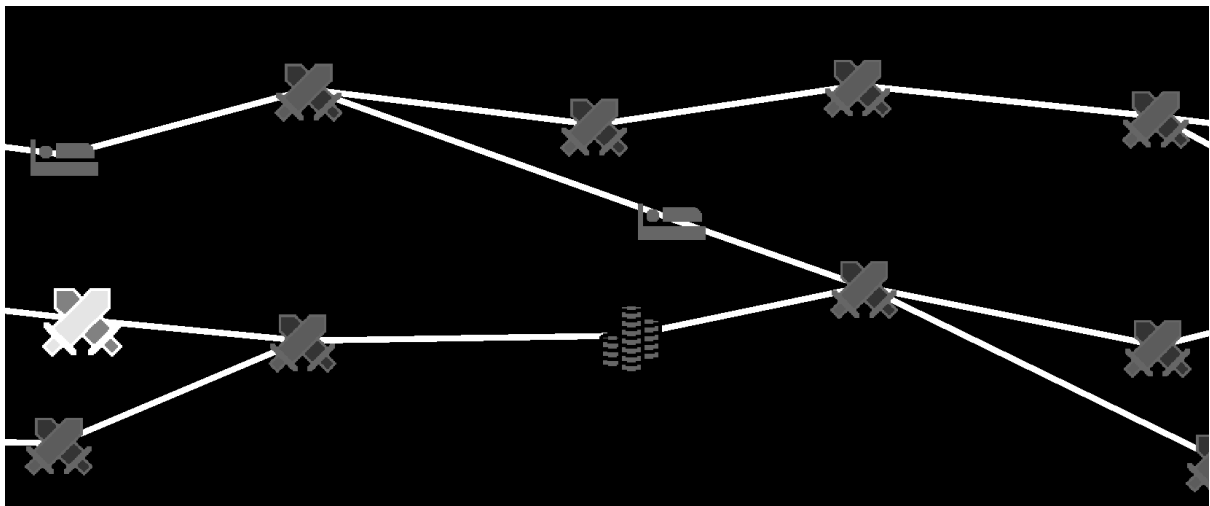
Within the game, the map is randomly generated, and different events are placed throughout. Before looking into how to create my own map.

I knew early on I wanted a randomized element within the game. As on smaller scale projects a random element can help to increase longevity. Or add unexpectedness and challenge in the game. Many games do not use true randomness. As it would quite difficult to randomly create a game.

Roguelike games such as the Binding of Isaac, FTL, Rogue Legacy and others usually have pre-made events and pre-made rooms that are randomly stitched together to create a cohesive package. Some event may also have randomness built in, so even if you get the same event the outcome would be different.

This is the model I wanted to go down when building the game.

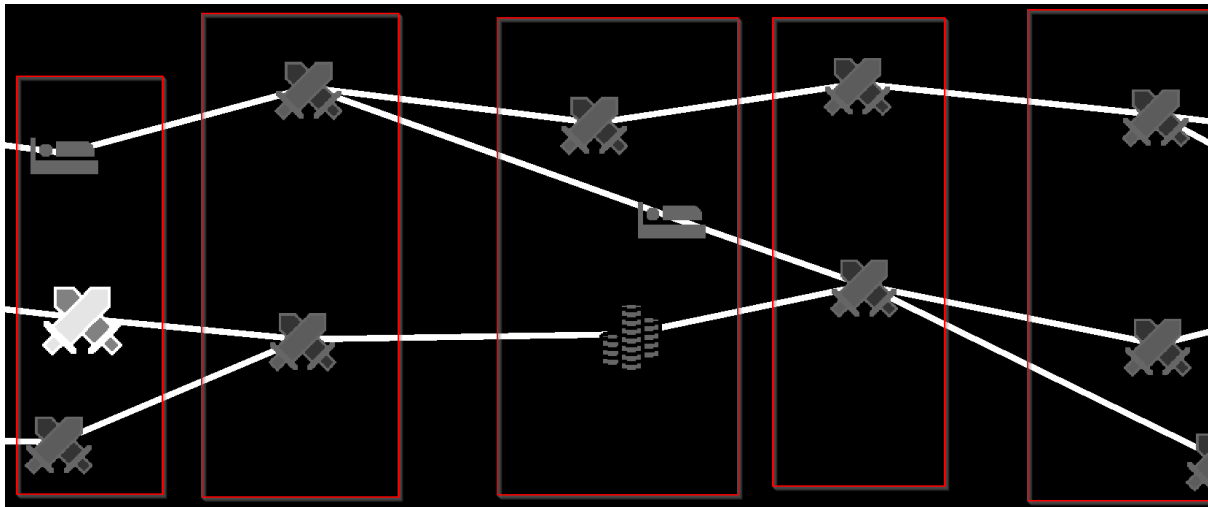
This is a snippet from a generated map:



As you can see, all points link together. And there are different symbols which refer to different event types. Swords – Battle, Bed – Rest, Stack of Coins – Shop.

When I went into this I figured each place you go to could be considered a ‘level’ of sorts. This would mean in terms of balancing the game. I could say that when a player reaches the final boss, they have progressed through ‘10’ levels. Using this you could assume certain things about the status of the player. Such as, they had completed at most 10 battle events. So, at most they should be ‘this’ powerful.

The map generator, given a number of levels, creates that many number of sections.



Within those sections based on the number of variables such as, how many nodes are there? What is the minimum spacing between nodes? Positions are generated.

Once those positions are created, each node within a section on the left. Looks to the next section on the right for a node it can connect to. (Usually the nearest).

After this, any nodes on the right that are missing connections then look back over to the left to find where they can link up.

They do this while also avoiding creating any overlaps, as the lines should not cross each other.

Once this has been done, nodes are then 'flipped' into special events. Such as shops or places to rest. This is because special events need to be spaced out. On top of this, to avoid repetition of special events. Before a node is flipped it looks, at it's parents and successors (based on the line connections) to ensure that for at least two levels, there are no repeated special event types.

Now this generator is also, not truly random. As there is a fixed number of special events that need to be placed within the map. This is to ensure players don't end up with no shops or no rest points.

This map is the core of the game and, as such has one of the more robust tests in the game.

```
@Test
public void simpleMapGenerationTest() throws Exception {

    int generations = 100000;
    float failCount = 0;

    for(int i = 0; i < generations; i++){
        try {
            MapGenerator mapGenerator = new MapGenerator();
            mapGenerator.generateGameMap();
        } catch(Exception e) {
            failCount++;
        }
        System.out.println(i);
    }

    System.out.println("Number of Generations: " + generations);
    System.out.println("Percentage Fail: " + (failCount / generations) * 100 + "%");

    Assert.assertTrue( condition: failCount == 0);
}
```

Once a map is generated it checks itself and then throws an error if the generation has failed.

The pseudorandom generation is not just limited to the Map. The events themselves also have these elements.

Places such as the shop, has a pool of items that it randomly selects from and tries to sell to the player. This ensures items can not show up twice in the same shop.

The 'Rest' event, is not random it just has a choice.

'Battle' events are random. But their randomness can be defined.

```
private EventCommand battleEvent1(){
    return () -> {
        return new BattleEvent.Builder(MapData.MAP_1)
            .enemyPool(EnemyFactory.FAST_BLOB, EnemyFactory.MAGE_BLOB)
            .primaryObjective(new DefeatAllEnemiesObjective())
            .bonusObjective(new CompleteWithinObjective(AbstractObjective.Reward.MORALE, rounds: 3))
            .build();
    };
}
```

Here is an example of creating a Battle Event within the game. The variable 'enemyPool' describes which enemies can be spawned within the event. The Map Data, describes which map is selected for this event. The objectives show which primary and secondary objectives will be used with this event, as well as the rewards you receive for completion.

The randomness for this, event is primarily in which enemies will be spawned and where they are spawned. As the map also contains spawn data.

3rd Party Systems

//Note: I planned to talk about using a map editor And it's benefits. But was unsure whether to mention it.

Critical Appraisal

Summary and Analysis of Work Completed

In all honesty I am lukewarm on the final state of the product. Maybe due to other ideas I now have in my mind, or because I feel it could be better. The game in my eyes could be better, but unlike other projects I don't feel like competing this one, once my dissertation is over.

The goal of using an Entity Component System to create finish skeleton game has been met, but I guess my lofty ambitions for it have not.

In terms of the original aims I've shifted from an action focussed game, to one where you must stop and think and decide how turns will play out. This shift in focus I feel was crucial to my opinions on the project.

This project ends with a game consisting of multiple characters, events items to buy in the store and in a playable state. But, I can't say it's in a state where people would want to play it. I feel issue with the project is that is it was 'creative' in nature.

What I mean by this is that, I wanted to investigate using an Entity Component System when creating a game. I also needed to create a game. However, these are two different problems and only one is truly relevant to my final grade. Regardless of what game I created what matters is the technical aspect behind it.

What I would have done in hindsight is perhaps pick a game to create a replica of. By aspiring towards something that already existed I would have had the goal posts in mind to where I needed to go to reach completion. I could have asked 'Does this look at play exactly like Pac-Man?' If the answer was yes, I would be done.

By picking my own game I found my original ideas were not fun. By opinions of different people and asking people to try out my initial ideas, I could see they weren't finding it fun. This kind of stumped me a bit since I needed to continue building the project, but I also didn't want to continue creating a game whose concept people weren't finding enjoyable. It was around this time I decided to try and switch my game over to strategy.

I switched the game to strategy because, one I'd never made one before and two I figured people who use mobiles usually play more strategic games as they demand less of your immediate attention. You can pause and think and then make a move. Your phone doesn't have to be in your hand permanently.

However, this open a new barrier that I didn't anticipate. Designing strategy is a rather challenging. Well, the way I went about it that is. The reason for this I found, is that I ended up in a bit of a chicken and egg type situation of what I needed to program first.

For example, currently, you complete battles and get rewards and then spend those rewards in the shop for equipment. For this you need to program the battles and then program how to generate

rewards and then program how to show these rewards to the players and then you have program the shop and the skills as well.

If you spend too much time thinking about the whole instead of just doing it in pieces, it can slow your progress because you are constantly second guessing what is the right way to do things. However, as I was new to strategy I was always thinking about how it would all come together. What was it that made it strategic? Where was the strategy?

I learnt this later than I would've liked but strategy is emergent with gameplay. Games aren't build around strategy, but strategy is created by the game. By this I mean looking at chess, there are a set of rules and from those rules millions of different games can be played. Those games were not planned by the creator of chess. The rules were. Players then went on to create strategies and didn't counter moves and such. Once I started implementing rules, the rest started to follow.

However, these rules were not my own, but inspired by a few games that I found when I was unable to build a game from which strategy could emerge. Which I guess is why I'm a bit sour on my game, I can't really claim it's design on my own thinking.

Another point, that I would have changed in hindsight, is I would've maybe use a couple of free art assets to help brighten the game a bit earlier in the project. As, creatively the game has ended up looking the same through most of the project. Quite dark and dingy. I know I stated I didn't really want to focus on art as this project was mainly focussed on the back end. However, like working on websites and adding some Bootstrap to make it look better can help change your perceptions of the work your doing.

It was difficult to show off my game to people who could test it, when it didn't look that great. Its visual aesthetics could already cloud their opinion and I'm sure mine was clouded as well. Compared to previous projects I rarely showed this one to anybody. Which is a cardinal sin among game developers. As you are missing out on crucial player feedback.

In regard to the general goal of the project. Which was creating a game using an Entity Component System. On that side I'm quite happy about having given a shot. Using this system feel pretty good, and I will go into more detail when I talk about my personal development.

Commercial and Economic Context

//Note: Not sure what this means or what this is about. I'm just following what's on the guideline.

//Some explanation would be most appreciated.

The methods and technologies I used such as LibGdx and Entity Component Systems are used to develop a number of games. More so, the ECS side. Although relatively new in the grand scheme of games, engines such as Unity have started to bake in components into their design. The idea of building things via components is certainly becoming more popular.

Other technologies I used such as map editors and storing skill information within JSON files is also a common practice in smaller projects.

My current project is not relevant on the grand stage as it is simply a mere hobby game. There are many out there and some can rise to prominence, but most go unnoticed. The game industry is a 'creative' industry. So like things like film, books, tv, music, etc. There are many who strive to be noticed, but only few can be.

Games are newer medium and have spread across the world, it's funny because many times when I said I was making a game for the phone, people would always reference that it might be the next 'thing'. It obviously won't be, but people seem to recognise that sometimes, some game do get lucky and do get notice and have the ability to be played by people everywhere. It's a newer medium of entertainment and like film, may become a main stay for a long while.

I myself will most likely, not be there. As I'm not a fan of the industry and like the idea of just making a few hobby games to help pass the time. Since I've found they can be useful in improving your programming a tiny bit.

Personal Development During the Project

In comparison to the summary and analysis this will have a much more positive spin. I've learnt a lot of this project. Most of it was caused by the fact that I went out of my comfort zone and decided to try and make a strategy. Deciding to create the game I ended up with was a weird double-edged sword. As below is a list of things I needed to learn, simply to make it through the project:

- Utility Theory when creating AI
- A* Pathing algorithms
- Creating my own Action Queue System
- Creating my own map generator
- Creating UI using LibGDX's Scene 2D UI classes
- Learning how to create Emergent Strategy
- Creating skill via JSON and uploading them into the game
- Learning how to use the Observer Pattern
- Creating Maps using 3rd party software and uploading it into the game

This is all without mentioning the title card of the project which was the Entity Component System. This whole project has been an iterative process of me trying something, asking myself if maybe there is a better solution. Researching and then implementing anything better that I've come across.

The only parts I disliked about the project were more on the creative side when I couldn't think of the correct direction for the game. Once I had a direction I found it very interesting to work towards the goal. It is incredibly refreshing when you find a solution for something and it feels like a good solution.

An example for this would be how I first tried to implement skills. It was awful. I had a class for each skill and code was being repeated, it just wasn't a good solution. I eventually distilled it all into one class that could be built. Then I found out that most game developers use XML and JSON to quickly describe variables that make up a skill/item. The game reads it in and boom, you can create skills using JSON and quickly edit damage values or what the skill does inside of the JSON file without having to look through a lot of classes.

In future I will want to use this technique for more than just skills. It could be used for enemies, items, even players or objectives. It opens your code to even be used by people who are less technical. If they want to change the image of a skill, just go to the correct part of the JSON file and change a variable. No need to look into any of the code.

However, although learning the patterns could be useful in future career opportunities. My experiences using LibGDX might not be as transferrable. I can't make native android apps very well, nor is my web design any better having done this project. These are common areas developers head towards. I'm not sure if employers will care that I've made a video game, when I'm unable to do the skills they find useful.

Conclusion

To conclude, this project was split into two halves. The half where I was learning different aspects of game development as well as how to properly utilize an Entity Component System. The second half was designing the game itself.

I enjoyed the former more than the latter. My original idea lacked fun, and with that I needed to come up with something new, which made progress slower than expected.

However, I don't leave this project too disheartened as I was able to reach the objective of having a functional product that can be played to completion by the deadline. Which is the main objective I wanted to hit. As it meant I was able to reign in my scope and stick to a deadline.

I can say that at the end of this project I have been converted into the idea of using Entity Component Systems in game development.

Bibliography

- [1], Rasmussen, Jakob. "Are Behaviour Trees a Thing of the Past?" Gamasutra Article, 4 July 2016, www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php.
- [2], Mark Dave, and Kevin Dill. "Improving AI Decision Modelling Through Utility Theory." GDC Vault, 2012, www.gdcvault.com/play/1012410/Improving-AI-Decision-Modeling-Through.
- [3], SolarChronus. "A* Pathfinding Tutorial." YouTube, YouTube, 28 July 2012, www.youtube.com/watch?v=KNxfSOx4eEE.
- [4], Patel, Amit. "A* Comparison." Introduction to A*, 2017, theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html.
- [5], Martin, Adam. "Entity Systems Are the Future of MMOG Development." T-Machine.org, 3 Sept. 2007, t-machine.org/index.php/2007/09/03/entity-systems-are-the-future-of-mmog-development-part-1/.
- [6], Papari, Adrian. "Introduction to Entity Systems." GitHub, 7 Feb. 2016, github.com/junkdog/artemis-odb/wiki/Introduction-to-Entity-Systems.
- [7], Papari, Adrian. github.com/junkdog/artemis-odb GitHub, 2016,
- [8], Gaul, Randy. "Spaces: Useful Game Object Containers." Game Development Envato Tuts+, 14 Jan. 2014, gamedevelopment.tutsplus.com/tutorials/spaces-useful-game-object-containers--gamedev-14091.
- [9], Apple Inc. "Entities and Components." GameplayKit Programming Guide: Entities and Components, 21 Mar. 2016, developer.apple.com/library/content/documentation/General/Conceptual/GameplayKit_Guide/EntityTypeComponent.html.
- [10] "Goals and Features." Libgdx, libgdx.badlogicgames.com/features.html.
- [11] Nystrom, Robert. *Game Programming Patterns*. Genever Benning, 2014.
- [12] Rabin, Steve. *Game AI pro: Collected Wisdom of Game AI Professionals*. CRC Press, 2014.
- [13] http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter17_Pathfinding_Architecture_Optimizations.pdf
- [14] <https://github.com/libgdx/libgdx/wiki/Scene2d.ui>