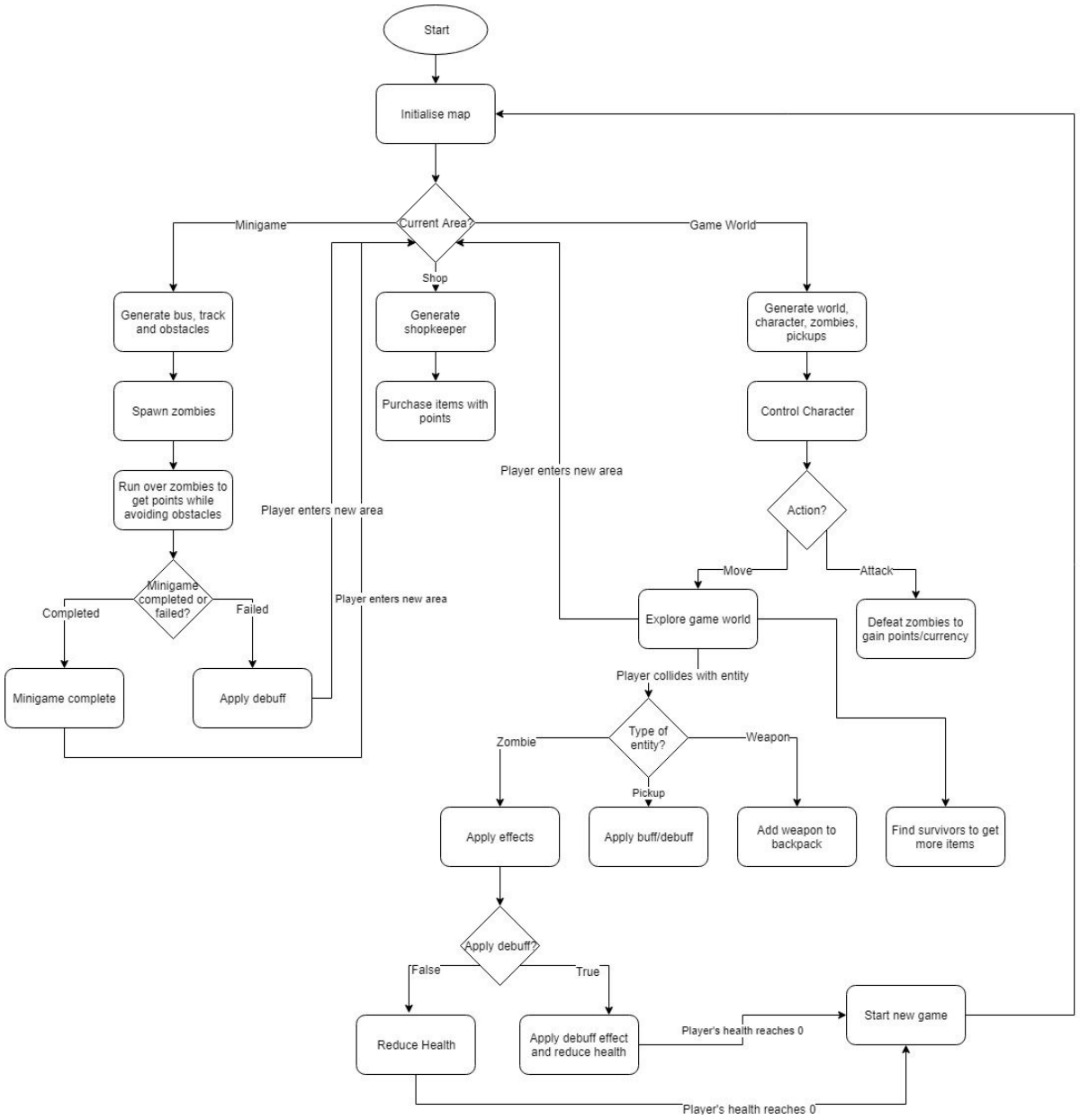
Architecture

Given the collected requirements specified in the above section, we have generated a High-level class diagram fulfilling these. A gameplay flow diagram outlining the general flow of the game and highlighting key features contained within the program, which will be implemented in a Java-based library LibGDX.

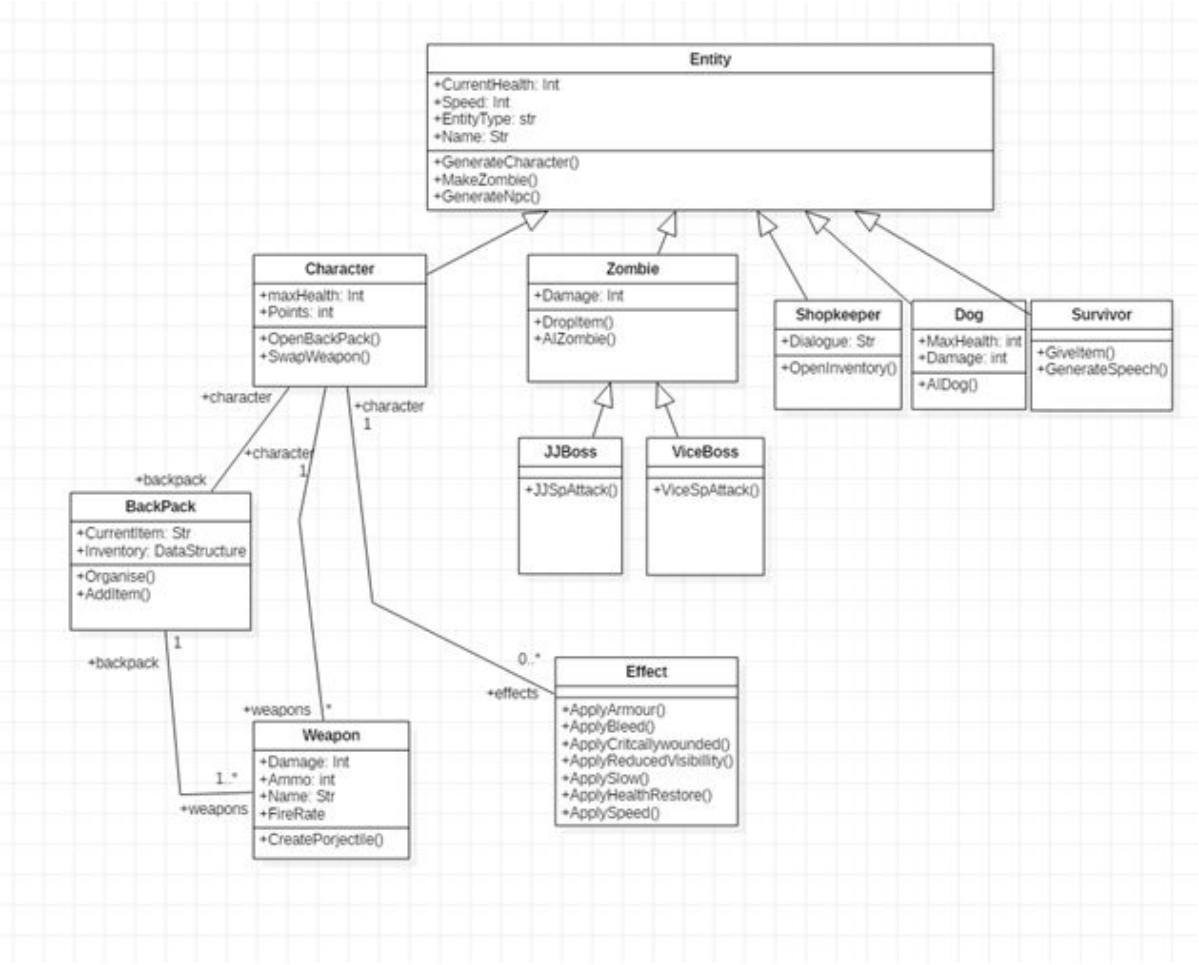
These models were constructed in StarUML and the online diagram drawing tool Draw.io.

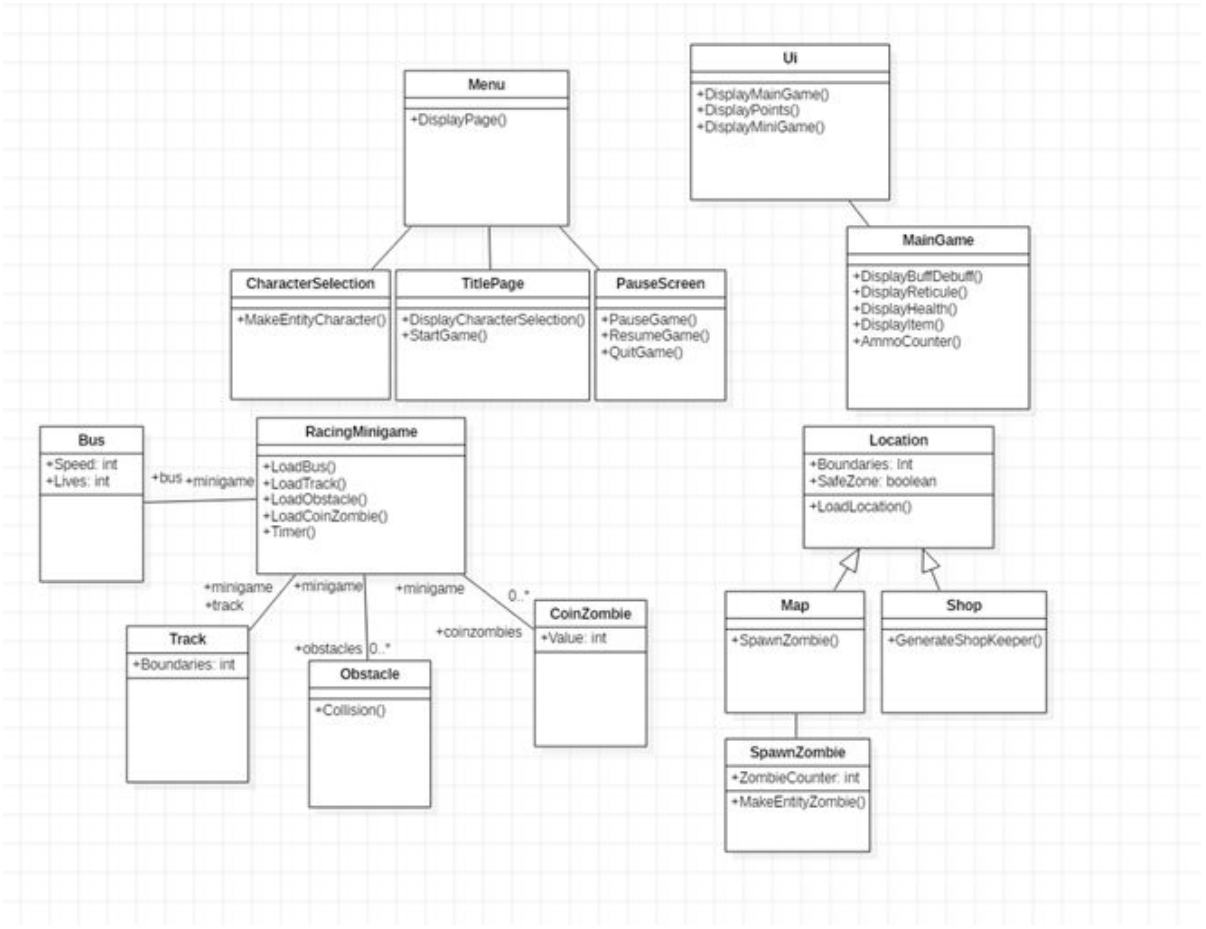
Game flowchart



To help with the design of the game and, in turn, it's architecture, a flowchart was constructed to show abstractly the basic functions of the game. It clearly shows various actions that the player can take, depending on the area of the game they are playing in. We found it useful when constructing the UML diagram, as it shows visually how different classes will perform together, showing how they relate in a more physical practical sense.
As the flowchart is abstract, it ignores certain elements that will be present in the actual game, such as menus, types of enemies and characters, and specific items and key presses. Instead, it just represents the general way the game will be played.
We decided to use a flowchart, as they are simple, quick, and easy to understand. Through a few pass throughs, one can easily see how the game will function while in play. We used draw.io, a free, web hosted piece of software, to create it. It is not the most professional tool possible, however it was simple and easy to use, fast, and required no software installation, so it was perfect for what we needed it for. It also allowed us to export the file as a JPEG, making it easier to document.

UML Diagram





When creating the architecture of the game, a UML (Unified Modelling Language) diagram was used to help with the design, we selected the UML program StarUML. This program allows the user to draw out their UML diagram in a user-friendly graphical environment. What makes it good is that it is convenient for the user to access the tools more easily to edit and assemble a representation of their project, with no programming needed it can be all be generated graphically. Most of the program requires little keyboard input, it is only needed to type out names and labels of things, so it mostly uses the mouse. The program comes with all of the necessary tools and mechanisms to create a fully fleshed out in-depth blueprint of your system. Some other reasons we chose StarUML is that it has the capability for asynchronous model validation, so it checks through the model for any mishaps in it. The software has support for code generation and model-driven development. Finally, the program is widely used throughout versus companies in industry these include Apple, IBM and EA (Electronic Arts) to name a few. You can see that it has a large variety of good aspects which make it a strong contender for our project.

For the construction of our UML diagram, we methodically went through the requirements, with a fair amount of consideration between us we devised a diagram to fit the proposal. First, we omitted functionality and connections between classes making everything a class from the requirements. By adding essential attributes to each class, we could see potential connections/similarities among the collection of classes. For example, the effects class was an assortment of individual classes which we saw to combined into one. In addition, we removed some associations which would have crossed the diagram to stop it from looking messy, but we tried

to keep most of the most important ones left in. Furthermore, to help us in the future we decided to keep all classes, functions and attributes all public so that we are not restricted when it comes to the actual programming construction of the software.

The table below will go into details about different parts of the UML diagram.

Entity	This class is to manage any humanoid-like being within the game holding some shared attributes that each of these entities e.g. Name and Health. It also has the functionality to make new entities which are its children classes. Its children classes include Character, Zombie, ShopKeeper, Dog and Survivor. This parent class was made to minimise the amount of duplication of attributes between classes to stop any confusion or conflicts and any unnecessary code to implement the game.
Character	This is used to manage the player by inheriting attributes from the Entity class, but it has its own specific characteristics such as maxHealth and Points. Plus, it has its own functions involving inventory management and weapon use.
Zombie	Like the Character, it has its own attributes. And the functions to drop items and attack the player. But it has it subclasses which are the game's bosses.
JJBoss/ViceBoss	Classes have their own combat AI but retaining the same concept of the Zombie class.
ShopKeeper/ Survivor	Classes are used to make the NPCs within the game for the player to interact with. Both having certain dialogue interaction functions. With all Shopkeepers having a fixed dialogue, but the Survivor having a range of dialogue choices. Both classes have inherited attributes from the entity parent class.
Dog	The dog companion has its own AI and own attributes with the ones given by the parent class.
BackPack	This class is used to house the items the player has enquired over the game and manage them; hence it is associated back to the Character class. It will have its own attributes and functionality including a data structure to allow for item storage and management. The data structure yet to be decided. It also has a connection to the Weapon class to allow weapons to be in the player's inventory, consequently, it has the attribute current item to grant this.
Weapon	The Weapon class is like the Backpack in that it has an association with the Character class. But it has the connection with the BackPack class to allow the player to use weapons that are in their inventory. It has its own attributes which grant it to have the stats of the weapon in use and by using its functionality it can create the projectiles needed.
Effect	This Effect class is used for handling the effect status of the Character, with its functions which are used

	to manage each effect. This is left separate from the character as it will need to interact with the UI to allow some of these effects to be implemented and it to display the icon of the buff/ debuff.
Location	This class is used to select the current location. Its attributes are used to confine the selected area and decide if the area is allowed to spawn zombies. Plus, it has the function to initialise a location. It is a parent to the Map and Shop classes.
Map	The Map class is to fully define the layout of the area and has the association to SpawnZombie so that it can put zombies into the game.
Shop	It has a predefined layout and will use the function to bring a shopkeeper entity into the game.
SpawnZombie	SpawnZombie is to control the creation of every zombie throughout the game using its function to create zombie entities. And by having the zombie counter attribute to regulate the number of zombie entities in the game.
Menu	The class created to manage what page is to be displayed. Has associations with CharacterSelection, TitlePage and PauseScreen.
CharacterSelection	Allows the player to choose the character they wish to play and then using its functionality to create an entity to make the character.
TitlePage	This is used to be the first thing the user interacts with. Its functionality to start the game and allow the continuation to the character section.
PauseScreen	This class is used to pause the game and its functions (pause/resume/quit game) this grants the player these actions.
UI	Used to manage in-game displaying graphics/interfacing objects. Like the Main/Mini game and points (used in both mini and regular game). It associates with the MainGame and MiniGame.
MainGame	This manages all of the displaying attributes for the main game through its functions. Example Health and Ammo Counter.
RacingMiniGame	Manages the Minigame aspect of the game. It has functions related to all of its associations (Bus, Track, Obstacle, CoinZombie).
Bus/ Track/ Obstacle/ CoinZombie	Associated with the MiniGame Class. Generates and functions to its respective class.