DESIGN DECISIONS

To make our design more efficient and flexible, we underwent a few decisions during the process of creating our design.

DEPENDENCY CONTROL

We allocated each member a portion of work in such a way that they shouldn't need to collaborate code until merging is required. This was done to reduce the dependency of members between each other and make the whole process more seamless. Thus, sub-systems could be created independently of each other, allowing us to pick up and fix errors sooner because these systems could be tested with the base game as soon as appropriate. For example, the Item and Grunt sub-classes could simultaneously be tested because they were coded separately from each other. Respective git branches were used to further enforce this separation.

Also a part of dependency control, connascence had to be minimised as much as appropriate.

If not necessary to be public, functions and attributes were made private in order to reduce the amount of methods having a scope beyond their class. An example of this can be seen in the PlaceBodyAction through the playerHasItem method. This helped make the design less mixed together and more divided into sections.

ENVIRONMENT OBJECTS

In the initial concept design for this project, the environment class wasn't added, instead the ground class was used as the backbone for all map exclusive objects as it was how the engine operated and we need to prioritise working with code that is functionally correct already. This has been rectified and improved in the current release.

ITEMS

Item sub classes are no longer being used, instead all item objects will be of the item class solely. This decision was made because all of the inventories instantiated by the engine

were for item types only, it would prove immensely difficult to implement item sub-classes inefficiently and effectively as it would likely require extending on many facets of the engine classes or hard coding it into the driver. Instead, each item would be identified by their character type (For example, ‘e’ for engine), which is implemented through any actions that require certain items to be used.

Because items will not have sub-classes, this means that keys will not be unique. Thus another decision was made to make any key open any door. The player only needs to have a key in their inventory in order to be able to open a door.

GROUND SUB-CLASSES

Each ground sub class will have their associated actions which are made allowable to the player in the menu when they are adjacent or standing on top of the ground. Whether the player is actually able to complete that action is decided within the action itself.

- **Door** - Represents a locked door.
    - **UnlockDoorAction**
- **RocketPad** - Ground that represents the rocket pad of which the rocket parts will be placed.
    - **PlaceBodyAction**:
        For this to work correctly an interaction between the inventory of the player and the location of the rocket pad is commenced where the specific item (In this case the rocket body part) is searched for by checking the representing characters for the one associated with the desired item. When successful, and the other rocket part (In this case the rocket engine part) is already in the rocket pad location, the rocket pad will be replaced with a rocket.
    - **PlaceEngineAction**:
        Serves same as the PlaceBodyAction but utilised for the rocket engine

part instead. It was decided these actions would be seperate, so that they would both appear on the menu at the same time.

- **Rocket** - Ground type that represents the completely built rocket.
    - **EnterRocketAction** - Initiates the victory condition of the game.

ACTOR SUB-CLASSES
(And associated actions/behaviour either made allowable to the player or NPC)

- **Grunt**
    - **FollowBehaviour**

        Basic follow behaviour which is inherited by the Goon subclass - there is no detection mechanic, the NPCs with this behaviour will attempt to move directly towards the player per turn. The game map has many walls and structures on it for the purpose of making evasion easier - if a hostile NPC is trailing the player closely, they may be split off by the player walking closely by walls and impassable structures

- **Goon**
    - **InsultBehaviour**

        Insult behaviour simply displays a chosen insult onscreen for the user on trigger. One side-effect of the method of its implementation is that the Goon class will stop moving towards the player for one turn if delivering an insult. This is useful as a common NPC AI pitfall is that hostile NPCs with followBehaviour will often ‘stick’ to the player, such that the player cannot escape them.

- **Ninja**

- **NinjaBehaviour**

  Simple evasive behaviour triggered when the player is within 5 units of the ninja.
- **StunBehaviour**:

  The stun behaviour of the ninja is unique in that it requires minor modification to the core Player class to function - the three key modifications made were to add an integer counter for how many turns the player is still stunned, a method executed during the deployment of menu options to skip player turns, and an interactive addition which allows the player to step through these missed turns even though no environment-affecting input is actually taken

- **Q (Qnpc)**

  - **RandomWalkBehaviour**:

    Picks a random, valid, cardinal direction of the four available to Q and moves to it 60% of turns. An improvement to the current implementation of this method would be to have a more random selection of the available directions, as the current decision method favours directions in clockwise order from North
  - **GivePlanAction**:

    Requires a special interaction that is initiated by the player (Instead of the NPC as done in most other actions) during the running of the game, thus for this action (And TalkToQAction) it was necessary to override the allowable actions such that whenever the player is adjacent to Q, the action will become available.
  - **TalkToQAction**

- **Miniboss**

  - **ShootLazerAction**:

    Attack that does moderate damage (10 HP) from a distance.
  - **LightSaberAction**:

    Attack that does high damage (15 HP) when player is adjacent. If possible the miniboss will always attempt this action first because it was

decided that it would be inefficient for the miniboss to choose to shoot their lazer at close range or do nothing when they are capable of dealing a great attack to the player.

CHALLENGES

Ending the game

An issue occurs whenever the player is removed from the map during a turn where a NullPointerException is thrown. It is believed this is caused because the other actors which still exist on the map try to make an action against the player, but instead cause a crash when the player reference points to nothing. Attempts were made to avoid this error by adding checks to each of the behaviours/actions (which utilise the player reference) which prevent them from being activated if the player doesn't exist. However the problem still persisted and a solution was not found. It was decided to leave this in because it doesn't inhibit the ability for someone to play this game.