Implementation

Team: No

Team Members:

- Jack Burman
- Sam Churchill
- C Lloyd
- Sam Ralph
- Rebecca Wardle
- Jiacheng Wu

Previous Group

Team: Vega

Team Members:

- Chloe Wardle
- George Grasham
- Lewis McKenzie
- Matthew Rogan
- Haopeng Zhu
- Benjamin White

NEW:

Demo Mode Structure Changes:

To implement a demo mode, several changes were made.
Firstly, the Character package was broken down into Entities and Movement packages. This allowed the movement system that is applied to an entity to be decoupled allowing for easy switching. This meant that the Player class could utilise either keyboard based movement or the pre-existing “ai” movement system.
This was achieved by renaming the Character class into the abstract Entity class , which the Player, Enemy and Npc classes all extend. And all movement based code was refactored out of Character/AiCharacter into Movement, UserMovement and AiMovement.

Hence, the Entity class now has a Movement parameter, which must be filled by either UserMovement or AiMovmement, as the Movement class is abstracted.
The names are pretty self explanatory, with UserMovement being controlled by keyboard input, and the AiMovement being controlled by the “computer”.
This change, apart from reducing code duplication, will also allow any entity to be controlled by any Movement system, for example the user controlling an enemy, and also allow new movement systems to be easily implemented.

The “enemies” container in EnemyManager, was also broken up into “activeEnemies”, “arrestedEnemies” and “jailedEnemies”. This was due to optimisation and readability, because it directly represents each state an enemy can be in, hence it is clearer to understand, and removes the need for checking the enemies state during an update.
It also speeds up the Player update() time in the demo mode, because only active enemies are being checked (to target), as opposed to every single enemy.

Furthermore, EnemyManager ownership was moved to Player, as it allowed for easier interaction between arresting/jailing and abilities.

Finally a new class GameDemo was created, extending Gameplay, which sets the Player movement system to an “ai” controlled one.

Save Game Structure Changes:
A new instance of Gameplay, called LoadGame was created, which loads a JSON file from disk, and builds all entities/instances from that file.
This caused the addition of a save() function to every entity/movement/ability class, and a new constructor using a JSON Object.
Finally, the world creation logic had to have a complete restructure, due to the order in which things were created causing severe bugs. For example, all teleportation and door creation must come after player creation. But due to super() constructors in Gameplay, this was not possible using the existing code base. Furthermore, by removing b2WorldCreation and breaking up game instantiation into multiple functions (buildEntities(), buildUI(), etc), it allows a customised version of the game to be built, so different features can be added/disabled as required.

Ability Structure Changes:
The initial Ability class was made abstract ("AbilityBase") thus removing the enemy abilities from the original class. The Law of Demeter was used in relying on abstraction without relying on concrete implementation. An interface was also created (IAbility) allowing for the use of a generic Host and Target. This allows both players and infiltrators to use generic abilities. The factory method was also used to generate abilities, where each type of ability extends from ability base. Hence allowing for the easy addition of new abilities later on. Furthermore, it is easy to modify in the future, without knowing too many details of the implementation. For example, it is easy for both player and enemies to use and disable abilities. When changing the concrete implementation, code users may not need to change the existing code just add the new code to achieve.In the abstract class, I put in the common implementation methods. For example, all abilities require cooldown and usage time, so it will be very convenient to reduce repetitive code.

Game Resume Changes:
Due to the previous groups use of lots of static variables, the game kept crashing when trying to start a new game (i.e. A game is won, and if you wanted to play again, you had to restart the entire game). A significant restructure of various data structures was required. This was also one of the reasons for moving EnemyManger inside the Player class.
In summary, almost all static variables were made non static, including the List of Doors and Systems and the requestSave function. However, due to the use of Gdx and the ability to obtain the screen from a static reference, a new function was created to obtain the correct reference. This reduced the need for significant changes to the game, but meant that all "target" generation on startup had to be removed. For instance, the enemy when created would have a system allocated. However this had to be removed because the Gameplay reference was invalid at that point of time (the screen returned was the mainMenu).

Unimplemented Features

The only feature that is not implemented, is difficulty modes.
This was allocated to a team member, at the very start of the project, but despite constant reminders, was not started.

Previous Implementation:

One requirement, elicited from one of our customer-team meetings, was that the game should have a demo mode. The demo mode, referenced in our requirements as NFR_DEMO, was initially planned to be implemented by recording an example of gameplay, and having this play for the user when the "demo mode" option was selected. However, due to time constraints, this was not included in our final version of the game and instead we had the game play out the same as in "play" mode however it is zoomed out so you can see what is going on all over the game. Meaning the requirement is not fully met but an attempt is made.

We had initially planned to include a mini map with gameplay; this is outlined in FR_MINIMAP where "a mini-map is displayed within the HUD, where the lead character can see his current location in relation to the entire map". As the mini map was difficult to implement and did not add much value to the game, given time constraints, we decided to remove this from the gameplay.

Finally, we also didn't implement a Tutorial mode meaning FR_TUTORIAL is not met. This is due again to time constraints and valuing a working implementation over a non-working one.