Architecture 2

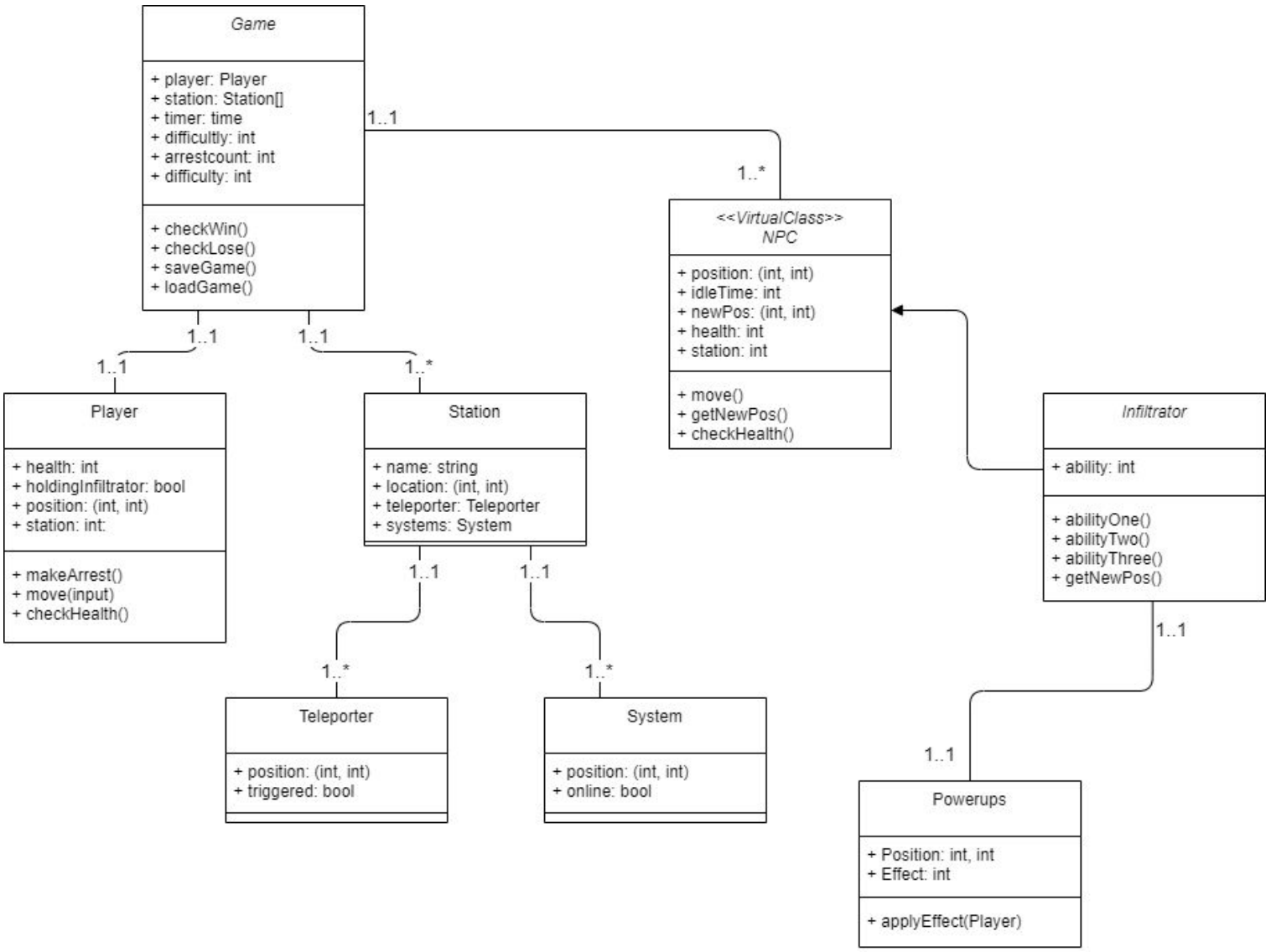
(Part of *Change2.pdf*)

Cohort 1, Team 4:

Robert Watts
Adam Wiegand
Josh Hall
Travis Gaffney
Xiaoyu Zou
Bogdan Lescinschi

a) (See full versions of these images on our website!)

Abstract Architecture - https://eng1-group4.github.io/Auber-2/assessment2/Abstract_Architecture.png



Concrete Architecture - https://eng1-group4.github.io/Auber-2/assessment2/Concrete_Architecture.png

Languages and Tools

To generate our abstract and concrete architecture we decided to use the Unified Modeling Language. We created class diagrams to plan what classes would be needed, the attributes of each class and the operations it can perform, and the interactions required between those classes. We chose this method because of its flexibility and readability. It is also the most ubiquitous design language used by software developers so should be readily understood by a wider set of people. In order to create these diagrams, we used the website draw.io because of its ease of use, and the features it provides whilst being completely free.

b)

Abstract justification

The abstract architecture is intended as a prescriptive tool to guide the implementation of the project. It gives an overview of what must be implemented without giving strict rules about how it must be implemented. The classes and methods illustrated in our abstract architecture reflect the requirements we have elicited from our brief and customer interviews. Design decisions such as infiltrators' special abilities and player's powerups are represented in the diagram, but without the decisions about the particular implementations of those abilities.

The diagram contains a low level of detail for a few different reasons. It allows us to capture only the most critical information at this early design stage to ensure we have the ideas for the game's core features. This allows for later expansion and additional features after the minimum viable product has been delivered. It also allows for flexibility in developing the concrete architecture if the implementation has to be changed due to limitations of the engine or evolving requirements. Our abstract architecture doesn't contain any language or engine specific methods so that it can be adapted to the tools and methods we decide to use for the project.

Concrete justification

For the figure, we can find the abstract design. That is the original thinking of the program and the basics of the concrete architecture. We can find that the abstract one only has 7 classes. When we work for the program, we add more classes and classify them into 5 main categories, UI, Screens, World, Entities and Pathfinding. We can find that we delete the Station class and change it to pathfinding and screen. NPC and player class are all included in the entities. Additionally, we added the world class which include the basic information for the map. This is the main difference of the two drafts of design.

UI:

- The UI group includes 2 classes- GameUI and Button. It provides the player HUD while playing by working with parameters of the game, and defines how buttons should function in the application.

Screens:

- Screens include 4 classes. MenuScreen shows what will be in the menu page, and it is the beginning page of the game. GameOverScreen displays when the player wins or loses the game. GameScreen displays when the player starts the game. LoadScreen displays the saving menu used to load into a previous game.
- Requirement FR_MENU [] is achieved in this part. GameOverScreen is to fulfill the requirement FR_LOSS_CONDITION []. LoadScreen fulfils the requirement FR_SAVING in a visual manner in order to aid the player.

World:

- The game world information is in the world class and used by the GameScreen page.
- This is to fulfill requirements FR_MAP and FR_MENU.
- It also contains all the constants that are needed when saving the game to a JSON file, which can then be loaded on another playthrough. Not only that, but there are also constants that can be altered according to the selected difficulty by the player which in turn make the game easier or harder.
- This fulfils FR_SAVING and FR_DIFFICULTY SELECTION.

Entities:

- GameEntity
    - This class contains all the information of entities that appear in the world. It includes the speed and max speed of entities, friction, velocity, collision and other information such as their position.
    - To fulfill requirement FR_HOSTILES and FR_SABOTAGE
- NPC
    - NPC class has two subclasses. The infiltrator class is the enemy which needs to be defeated by the player and attack the space. The civilian class contains the civilians. They do not have an impact on the game. They are designed to hide the infiltrator. The infiltrators and civilians look the same, but only the infiltrators will attack the space. To fulfill the requirement FR_ALIENS_COUNT.
    - These two classes can fulfill requirement FR_ALIENS, FR_ALIENS_COUNT and FR_HOSTILES_SPECIAL
- Player
    - The player class is the entity controlled by the user, which extends the GameEntity class. It contains the texture for the player and handles user input to control the player, and the player's interaction with other entities.
    - That is to fulfill requirements FR_ATTACK_NOTIF, FR_HEAL, FR_ARREST.
- Projectile
    - This class handles the projectiles that are shot by the infiltrators and handles collisions with the player, playing a sound if contact is made and applying a debuff (slow, blind, confused).
    - This class fulfils the requirements FR_HOSTILES_ABILITIES, FR_HOSTILES_ATTACK, FR_SPECIAL_ABILITIES.
- Powerup
    - The powerup class was added in order to facilitate the requirement FR_POWERUPS.
    - This extends the Game Entity class, and handles collisions with the player and the buffing effects of each of the 5 powerups- namely heal, shield, invincibility, speed, and telesplosion.
- The entities all include toJSON methods which shall return a JSONObject representing them, facilitating the FR_SAVING requirement, and allowing entities to be saved in whatever state they are found in.

Pathfinding:

- NavigationMesh
    - This is the map section of the game. It gives the player a reminder about the accessible path.
    - To fulfill requirements FR_TELEPORTER AND FR_MAP.
- PathNode
    - It gives the node pathfinding attempt.
    - This class can fulfill the requirement FR_PLAYER_TILES.