Software Requirements and Design Document

For

Group 15

Version 1.1

Authors:
Brian F
Cooper P
Chelsea W
Madison D
Richard S

1. Overview (5 points)

Give a general overview of the system in 1-2 paragraphs (similar to the one in the project proposal).

A 2D pokemon-like game with a mix of functionalities from different pokemon games using the FSU campus as the map. The game will include typical turn-based battles, exploration of a map, and catching pokemon.

The player will have access to the map in which the player will be able to catch different types of pokemon in different areas. There will be NPCs located around different areas of the map which players will be able to battle in order to get rewards and advance to further areas of the map. Battling will drop rewards such as in-game currency and items you can use to level up and ultimately evolve your pokemon.

2. Functional Requirements (10 points)

*List the **functional requirements** in sentences identified by numbers and for each requirement state if it is of high, medium, or low priority. Each functional requirement is something that the system shall do. Include all the details required such that there can be no misinterpretations of the requirements when read. Be very specific about what the system needs to do (not how, just what). You may provide a brief design rationale for any requirement which you feel requires explanation for how and/or why the requirement was derived.*

1. Users can use corresponding arrow keys to be able to move up, down, left, and right based on keyboard input. This is a very high priority.
2. In a pokemon encounter or battle encounter with a trainer, the user can select what items, pokemon, and pokemon moves to use on their turn. The status of the player and the pokemon or trainer's pokemon should update dynamically depending on actions. This is high priority as battling with pokemon is a vital part of the game.
3. Moving around in set locations such as tall grass will have a chance to trigger a pokemon encounter, this is a high priority since it is the basis of the game.
4. In a pokemon encounter, when selecting the action of throwing a pokeball, provide a chance of catching the pokemon in the pokemon encounter. If the pokemon is caught, add the pokemon to the current party. This is a high priority since it is the basis of the game.
5. In a trainer battle, the user will be able to select the action of switching pokemon, or choosing a move. This will continue until the battle is concluded since you cannot run from a trainer battle. This is a high priority because it is one of the main focuses of the game.
6. After winning in an encounter(capturing a pokemon, defeating all pokemon of a trainer, defeating pokemon in a pokemon encounter), the user will be rewarded with candy which can be used to level up their pokemon. This is a high priority as it allows for progression and challenges in the game.
7. Once a pokemon meets certain requirements(reaching a certain level), the pokemon will evolve to another form. This is a high priority as this allows for progression and variety.
8. Once unevolved pokemon reach certain, pre-defined, levels, they will be able to learn new, stronger moves. This is a medium priority because it will be necessary to progress in the game.
9. Sprite of player, npc's, and tiles of map should be visible to the user, this is a high priority as other functionalities rely on the user being able to see what is occurring in game.
10. Moving the player sprite into a trainer's line-of-sight will trigger a battle encounter to occur, this is a medium priority.
11. Menu screen should be the first screen to be displayed when the program runs and can select buttons displayed and be directed to an expected display, this is a medium priority as it provides the user some options before starting gameplay.
12. Users can click on a nurse npc to trigger dialog asking to heal pokemon, if yes is selected then should update all pokemon in the current party to full hp. This is a medium priority.

13. When exploring, user can access a menu that allows them to view the status(hp and status effects) of all pokemon in their current party and allow users to use items outside of battle(such as potions to increase a pokemon's hp, revive to revive fainted pokemon, items to heal status effects or boost some aspect of the pokemon). The pokemon should update to the appropriate state depending on the item used. This is a medium priority.
14. When planting seeds in a zone that allows for farming, time plants grow and updated sprite to indicate the growth and when the plant can be picked. When the plant is picked, it can then be used as an item. This is a low priority.
15. When the save option is selected, the user's pokemon, position on the map, items, etc. will be saved to the database so they can load back into the game right where they left off. This is a high priority so people don't have to reset their progress every time the game is exited.
16. Trainers should not be able to pass through Block objects (appear as rocks in game).

3. Non-functional Requirements (10 points)

*List the **non-functional requirements** of the system (any requirement referring to a property of the system, such as security, safety, software quality, performance, reliability, etc.) You may provide a brief rationale for any requirement which you feel requires explanation as to how and/or why the requirement was derived.*

1. The game should run at 60 FPS, this is for a smoother gameplay experience.
2. Should not take more than 20 seconds to load between game sections, where game sections include: a different or new map area, pokemon encounters, and battle encounters.
3. User input should take no more than 5 seconds to process.

4. Use Case Diagram (10 points)

*This section presents the **use case diagram** and the **textual descriptions** of the use cases for the system under development. The use case diagram should contain all the use cases and relationships between them needed to describe the functionality to be developed. If you discover new use cases between two increments, update the diagram for your future increments.*
***Textual descriptions of use cases**: For the first increment, the textual descriptions for the use cases are not required. However, the textual descriptions for all use cases discovered for your system are required for the second and third iterations.*
Textual descriptions of use cases:

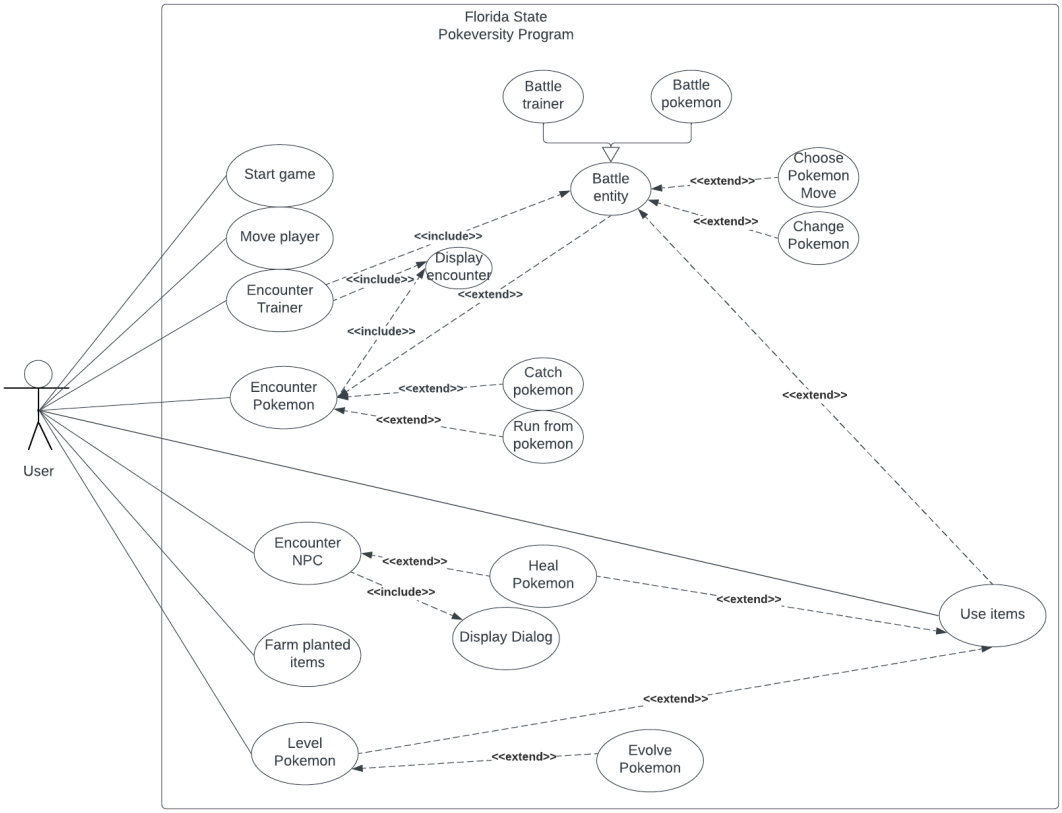
- Start Game:
  - Actor: user
  - Description: Game starts
  - Preconditions: All files and libraries used are obtained
  - Postconditions: Game window is displayed
  - Mainflow:
    - User types in the command to launch the main game
    - Loading screen shown to prompt user to enter game
  - Alternative flows: new game option, load game option
  - Exceptions: crash
- Move player
  - Actor: user
  - Description: player moves
  - Preconditions: Game is running
  - Postconditions: coordinates of the player changes based on directional button pressed.
  - Mainflow:
    - User presses an arrow on keyboard
    - Player sprite moves in the direction of arrow key pressed
  - Alternative flow:
    - Obstacle in the way arrow key is pressed

                - Use case ends with player sprite not changing coordinates
        - Exceptions: crash
- Encounter trainer
    - Actor: user
    - Description: user encounters NPC which prompts conversation.
    - Preconditions: Game is running and user has alive pokemon
    - Postconditions: dialog of trainer is displayed on screen
    - Mainflow:
        - the user gets engaged in conversation when moving the player sprite in line of sight of the NPC.
    - Alternative flow:
        - If user's player has no pokemon that can battle
            - Display dialog of trainer not triggering battle
        - If user's player has pokemon that can battle
            - Display dialog of trainer to start a battle

    - Exceptions: crash
- Encounter Pokemon:
    - Actor: user
    - Preconditions: Game is running and user has alive pokemon
    - Postconditions: pokemon encounter screen is displayed
    - Description: user encounters pokemon in tall grass to either run from, catch, or defeat pokemon.
    - Mainflow:
        - User moving player sprite through grass or areas where one can encounter pokemon
        - If pokemon chance is it, starts an encounter of a pokemon that can be encountered in the area
        - Pokemon battle begins.
    - Alternative flow: catch or run from pokemon
    - Exceptions: crash
- Encounter NPC
    - Actor: user
    - Preconditions: Game is running
    - Postconditions: dialog of NPC is displayed on screen
    - Description: user encounters NPC which prompts conversation.
    - Mainflow:
        - user gets engaged in conversation with NPC after clicking on them with mouse cursor.
    - Alternative flow:
        - If NPC has dialog option to heal pokemon and user clicks that
            - Pokemon healed
- Level Pokemon
    - Actor: user
    - Description: pokemon level raises by 1
    - Preconditions: Game is running and have candy in inventory
    - Postconditions: Level of pokemon increased by the amount of candy used
    - Mainflow:
        - user enters inventory
        - user selects item called candy
        - user selects pokemon to use candy on
        - the selected pokemon's level increased by one
    - Alternative flow:
        - Pokemon reaches a certain level
            - Pokemon evolves
    - Exceptions: crash

- Battle entity
    - Actor: user
    - Description: user begins battle
    - Preconditions: Game is running and encounter entity
    - Postconditions: Player is rewarded for a win or faints when user loses
    - Mainflow:
        - When user moves the player around
        - Player sprite triggers encounter
        - Player can select actions in battle
        - Give rewards based on win or loss
    - Alternative flow: either battle trainer or wild pokemon
    - Exceptions: crash
- Choose pokemon move
    - Actor: user
    - Description: picking move a pokemon uses in battle
    - Preconditions: Game is running and in battle
    - Postconditions: Pokemon uses move
    - Mainflow: select move option from menu in battle
    - Alternative flow: no alternative
    - Exceptions: crash
- Change pokemon
    - Actor: user
    - Description: user switches pokemon currently in battle
    - Preconditions: Game is running and in battle
    - Postconditions: Pokemon user is using is changed
    - Mainflow: current pokemon switched with another from battle party at users choice.
    - Alternative flow: no alternative
    - Exceptions: crash
- Use items
    - Actor: user
    - Description: user uses item on pokemon or user
    - Preconditions: Game is running
    - Postconditions: Item used will apply its effect
    - Mainflow: item takes effect on pokemon/user instantly.
    - Alternative flow: no alternative
    - Exceptions: crash
- Evolve pokemon
    - Actor: user
    - Description: pokemon evolves
    - Preconditions: Pokemon reaches target level
    - Postconditions: Old Pokemon will be replaced with evolved pokemon
    - Mainflow: pokemon evolves
    - Alternative flow: no alternative
    - Exceptions: crash
- Catch pokemon
    - Actor: user
    - Description: Pokemon is caught and added to party
    - Preconditions: Game is running and in battle, user has open spot in party
    - Postconditions: New Pokemon is added to the party
    - Mainflow: user uses pokeball to catch pokemon
    - Alternative flow: no alternative
    - Exceptions: crash
- Run from pokemon
    - Actor: user
    - Description: user flees battle

    - Preconditions: Game is running and in battle, user has at least 1 alive Pokemon
    - Postconditions: Dialog will be displayed and screen will switch back to the map
    - Mainflow: user exits battle through run button and goes back to map.
    - Alternative flow: no alternative
    - Exceptions: crash
- Heal pokemon
    - Actor: user
    - Description: pokemon healed
    - Preconditions: Game is running and Pokemon are not at full health
    - Postconditions: Pokemon will be restored to full health
    - Mainflow: pokecenter heals pokemon
    - Alternative flow: item used on pokemon to heal
    - Exceptions: crash
- Display dialog
    - Actor: user
    - Description: dialog shows up on screen
    - Mainflow: user interacts with NPC which displays dialog
    - Alternative flow: no alternative
    - Exceptions: crash



5. Class Diagram and/or Sequence Diagrams (15 points)

*This section presents a high-level overview of the anticipated system architecture using a **class diagram** and/or **sequence diagrams**.*

*If the main **paradigm** used in your project is **Object Oriented** (i.e., you have classes or something that acts similar to classes in your system), then draw the **Class Diagram of the***

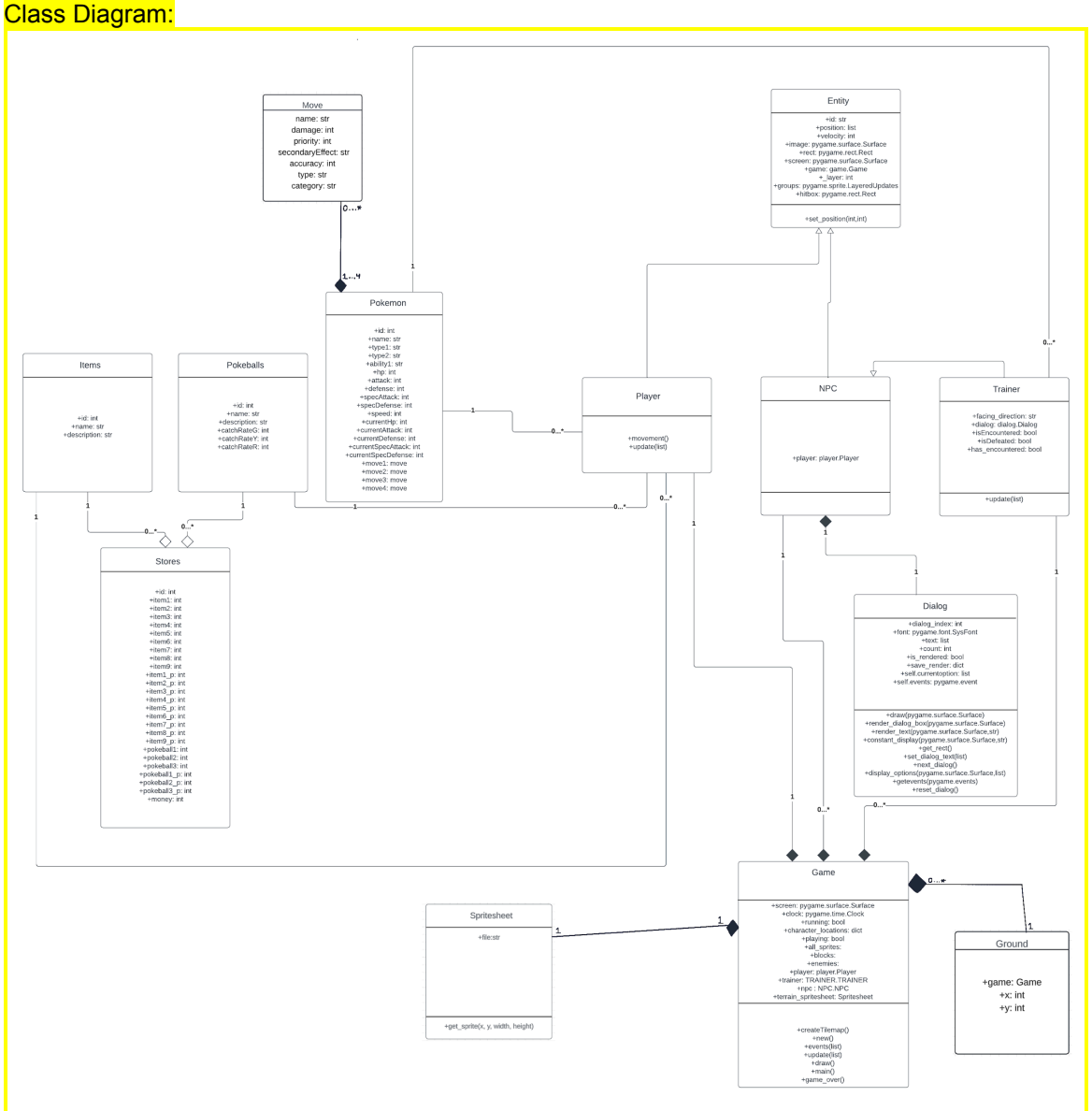
entire system and Sequence Diagrams for the three (3) most important use cases in your system.

*If the main **paradigm** in your system is **not Object Oriented** (i.e., you **do not** have classes or anything similar to classes in your system) then only draw **Sequence Diagrams**, **but for all the use cases of your system.** In this case, we will use a modified version of Sequence Diagrams, where instead of objects, the lifelines will represent the functions in the system involved in the action sequence.*

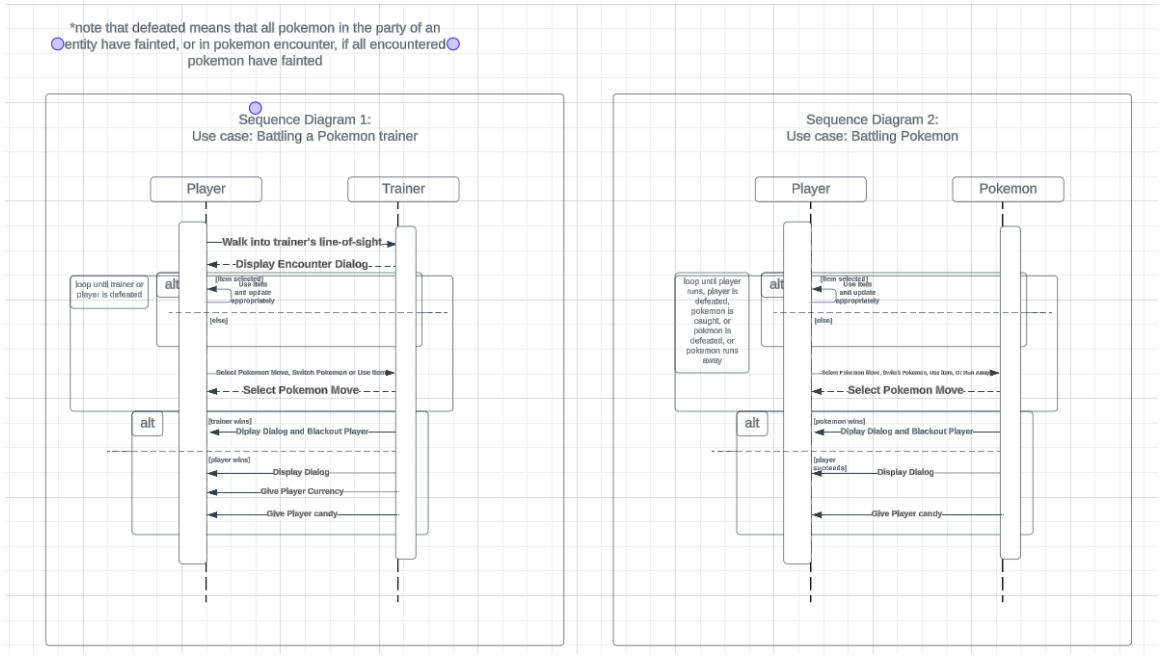
***Class Diagrams** show the **fundamental objects/classes** that must be modeled with the system to satisfy its requirements and **the relationships** between them. Each class rectangle on the diagram **must also include the attributes and the methods of the class** (they can be refined between increments). All the **relationships between classes and their multiplicity** must be shown on the class diagram.*

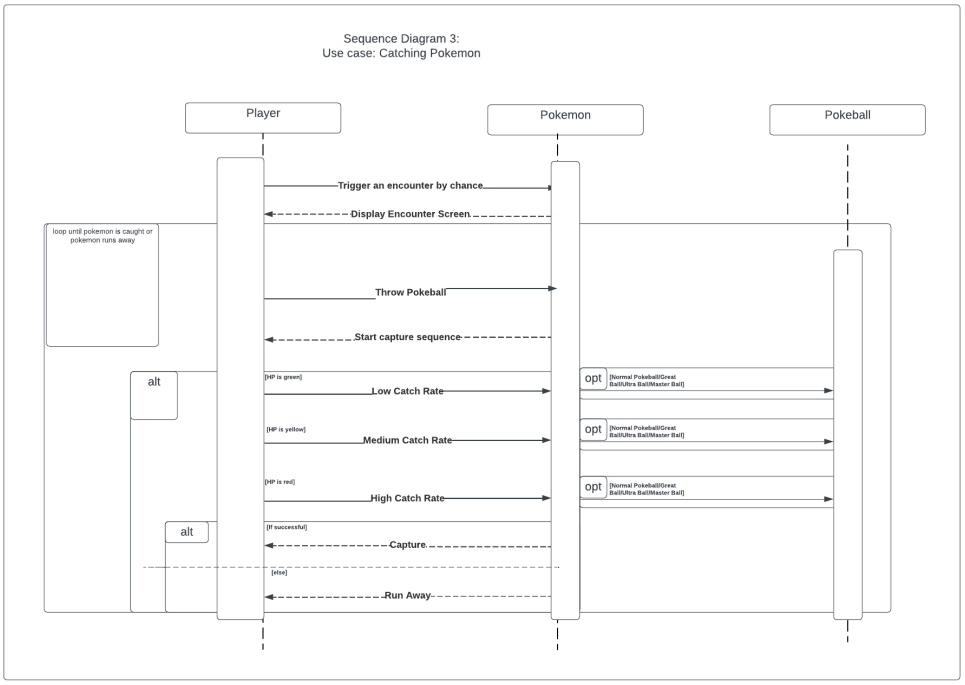
*A **Sequence Diagram** simply depicts **interaction between objects** (or **functions -** in our case - for non-OOP systems) in a sequential order, i.e. the order in which these interactions take place. Sequence diagrams describe how and in what order the objects in a system function.*

Class Diagram:



Sequence Diagram:





6. Operating Environment (5 points)

Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.

Our program is created to be run on the players' local device. The database is included in the program so that when logging in, the player does not need to connect to any server to access

their own data. The system running the game must have some version of Python3 installed in order to run the program.

7. Assumptions and Dependencies (5 points)

List any assumed factors (as opposed to known facts) that could affect the requirements stated in this document. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project.

Assume that the program is executed in an environment where Python(within some version 3)(programming language) and the Pygame API is installed.