GAME3023 Term Project - Pokemon-Style Turn-Based RPG Battles

Group size: 1-2
Weight: 20 pts to course grade, up to 150% out of 100% for 10 bonus marks

Summary:
Work **alone or in pairs**, using many aspects of the Unity Editor to create a **vertical slice prototype** of a 2D turn-based **Role Playing Game (RPG)**. In this game, you control a character exploring the world and resolving random encounters, similar to Nintendo's Pokémon series. (E.g. https://www.youtube.com/watch?v=C034iux-EJ8)

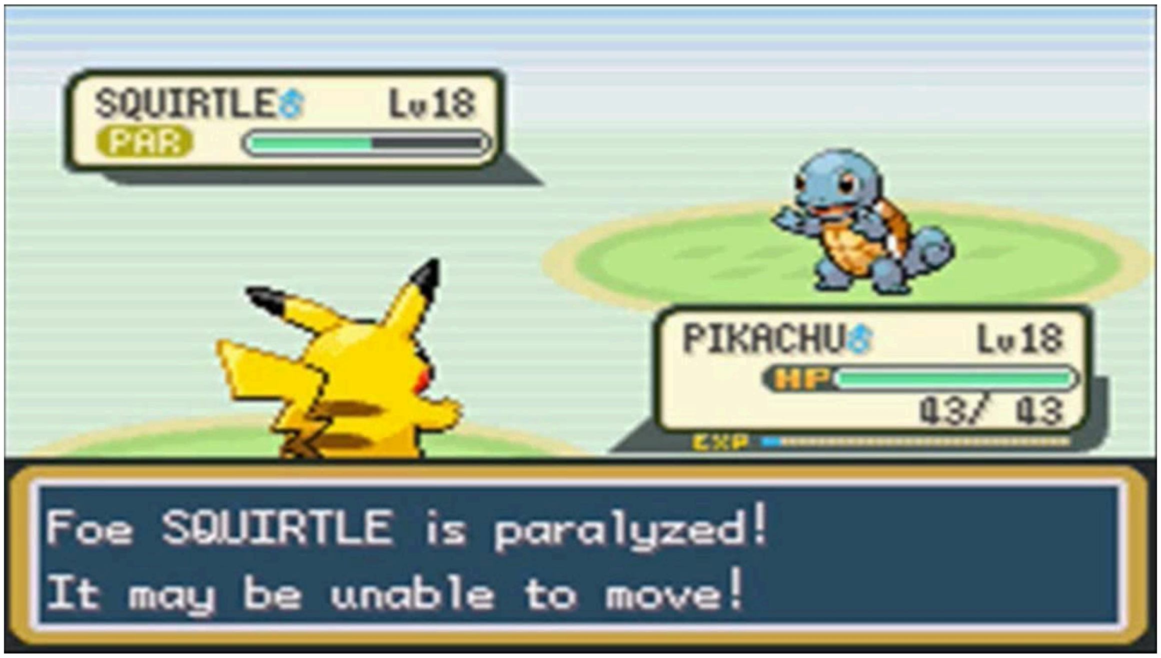

Figure 1: A battle from the Pokemon game series by developer Game Freak Co.

For this project, you and your partner may re-use code or assets which you have previously made for one of the course's Exercises!

Learning Outcomes:
Test a student's ability to exercise creative control, long-term planning/project management, to find and filter information, gather user needs, define goals &

constraints, design and implement scalable systems in Unity, cooperate with others, and communicate value.

1. **Identify and explain specific techniques applied to create enjoyable game experiences**
2. **Design and iterate on meaningful objectives, feedback, and actions to provide players with relevant information to make interesting decisions.**
3. **Develop specific mechanics and game design patterns relevant to the case study of 2D turn-based RPG style games.**
4. **Communicate design intent, process and value for end-users.**
5. **Apply 2D Unity game development skills.**
6. **Write robust, scalable code that supports design iteration.**

Mandatory Criteria
In this assignment, some criteria are marked as **Mandatory** which means each criterion marked as such must be completed, or you will receive a mark of 0 for the entire assignment.

Assignment Details

Source Code Submission* - Mandatory

- A private GitHub link to the complete Unity project. You must invite your professor as a collaborator.
- The commit history must show activity that indicates collaboration between members. All members must have a meaningful contribution.
- Ensure at least the **.vs** and **Library** folders in your project are in the .gitignore file and not included in the repository. These folders contain large amounts of auto-generated cache data and are not required when you share your project

Itch.io Game Submission* - Mandatory
Create an itch.io page, ensure it is marked as **published**, then submit the link. The page content should be as follows:

1. **How to Play:** includes a description of everything needed to play the game. **Controls**, **mechanics**, etc. Mention any special **cheats** included with the game... or bugs you would like to re-cast as "easter eggs".
2. **Credits:** shows the names and student numbers of the creators, as well as their specific contributions to the assignment as a whole. **If a student's name is missing**, they will receive a grade of 0
3. **Devlog:** A Youtube link to a **5 to 20** minute narrated **developer video log,** where you show off the game's features as it runs, and then describe how it was developed and tour the project. You do not need to repeat descriptions of any of the features already discussed in a Weekly Exercise, however you should **pay careful attention to the features in this assignment sheet e.g. Abilities, as well as any extra features which go beyond what is listed.** There may be more to discuss than you can fit. In that case, choose parts you are most proud of and speak on them. Both group members should be

identified and heard narrating content in the video. The time can be divided in any way, as long as each contributing member speaks for at least 1 minute.

4. **A WebGL build playable in browser, and a built executable for Windows 64-bit** uploaded via itch.io's releases section. You may include additional versions e.g. iOS builds if you choose.

*Afterwards, you may additionally add your page to an **Unlisted Game Jam** hosted for the assignment to share your games with others and have a bit of fun. Participation is optional and has no effect on grades.*

Credit/Acknowledgement of Sources of Ideas and Assets - Mandatory

- Sufficient citation of resources which helped you develop the ideas for code included in the project
- Sufficient credits and licenses for assets used in the project leave no doubt about the sources of content and the legality of their use in the project
- Include this as a text file within the project labeled "Licenses.txt" and as a submenu of the title screen. It should detail the licenses under which each asset or asset pack is published. An example is provided in the Lab repository.

Cheats

- To ensure the entire game can be easily tested, it is recommended to add cheats for various actions e.g. helping to win/escape encounters, gain resources, etc. List them in the game release page

Menus/Options

- A title screen with the game title, a New Game & Continue button that transition to the **Overworld Scene** and a quit button to exit the game
- Add credits for your name(s) and student ID(s), as well credits and licenses for assets used in the project
- Options to modify SFX volume and MUSIC volume independently

Overworld Scene

- A small explorable 2D game world (e.g. ~30-60 seconds to walk across from one end to another)
- Correctly layered tile map with an appealing composition of tiles
- Interesting lighting
- Correct level collision (nothing unexpected)
- Fun design to explore and/or interact with

Player Character

- Player character that can move around and collides with obstacles
- Animation states and transitions that respond to movement using the Unity Animation Graph

Encounter System

Is there purpose and variety in the conflicts of the game? Are there meaningful decisions and planning for the player related to this conflict?

- While in the Overworld Scene, the player may enter a Random Encounter while **moving** (not just standing) over a **Tall Grass** sprite
- Encounters use a turn-based menu battle system. The player acts, then the opponent acts, etc.
    - One opponent faces the player on the opposite end of your screen
    - On the player's turn, there are four options of **Ability,** which appear as clickable UI buttons showing their name
    - Choose menu options to perform actions implemented using UnityEvents. When your action is over, the opponent's turn begins and they choose an action as well
    - On the opponent's turn, they also choose an Ability from their own set, just like the player
    - There are both win and loss conditions for the Encounter
- The game should clearly indicate victory or defeat after each Encounter
- Upon losing, the player resets to their last Save

Ability System

Do the underlying systems allow designers to be free to make and balance many interesting gameplay effects with minimal extra programming effort?

- In an encounter, the player can use **Abilities,** which can apply effects to a character in the battle system
- Each Ability triggers a different visual effect, sound, and animation to differentiate them
- The character should be decoupled from Abilities. In other words, any character can have any set of Abilities
- There should be at least **four** unique abilities in the game. *For example:*
    - **Escape --** has a X% chance to either succeed, or do nothing, passing the turn over to the opponent. Provide feedback on success or failure of the ability
    - **Struggle --** has an X% chance to instantly win the battle
      -The player must start with at least these two abilities. Provide feedback on success or failure of the ability
- Upon winning an Encounter, the player can **gain an ability**
    - You can do this in different ways – take one from the opponent, randomly from a list, or even let the player select one from a skill tree if you are bold
- Consider carefully how this system may scale to a long-term project which requires multiple designers maintaining over 100 unique Abilities. In Unity, Prefabs and/or ScriptableObjects could help to define these.
- The more flexible and unique each **Ability** can be, the better. It is more interesting (and difficult to code) if each Ability does completely different effects than if every Ability is functionally the same, but with a few different numbers

Achievement/Award System
Adding extra objectives to a game for players to achieve can acknowledge and reward players or motivate them to try features of the game they would not have otherwise. Can you build a system to do this, which interacts with many parts of the game, without accidentally creating a codebase with the architecture of a spaghetti bolognese?

- An in-game Achievement screen tracks progress in completing specific objectives, as well as which have been earned and which have not. The screen should display the criteria of each objective e.g.
    - Take 10 steps in the Overworld
    - Open the Achievements Screen
    - Use _____ ability on _____ enemy 3 times
- There are achievements connected to many game features. They are not limited to one area of the game and can have criteria which are very specific and hard to accomplish. E.g. achievements for time played, locations in the Overworld, owning certain abilities or combinations of abilities, executing specific sequences of actions inside an Encounter
- When a player earns an Achievement, a popup appears to acknowledge what the player did.
- Use Event-based programming/Observer Design Pattern to allow the Achievements system to track progress on meeting conditions without forcing the programmer to insert achievement-related code into many different unrelated parts of the game.

Encounter Opponents
Can the player feel that they are against an interesting and challenging foe?

- Multiple types of opponents possible with different sets of abilities. They do not all need to have four abilities. For example, one enemy may have Abilities *Scratch* and *Bite*, while another may have *Bite* and *Growl*
- AI uses the same ability system as the player and follows all the same rules. An enemy may have any ability the player can have and vice versa.
- Encounter opponents are controlled by an AI script which is more predictable than random, and less predictable than using the same move every time
- If using the same set of abilities as the player and having the same stats, an AI should be smart enough to present a threat to the player

Game VFX/SFX
How good can you make this feel?

- Appropriate SFX and VFX (UI sounds, footstep sounds, particles, screen shake etc.) should punctuate actions and provide feedback
- A Music system should play different music inside and outside of battle
- Audio should be smooth. There should be no “popping”, sudden cuts, or other audio issues
- Game should have solid thematic coordination and nothing should seem *too* out of place

Saving/Loading

How forgiving is the game for players over multiple sessions?

- At any time outside of an Encounter, the player can save their game by clicking a UI button with the mouse. This save includes all of the player's current state including abilities. **Be sure to show this in your video.**
- Upon losing in an Encounter, the player is reset to the last time they saved
- When pressing the "Continue" button in the Title Scene, the game automatically loads the last Save

Overall project and code organization

How well does the project fit together? How hard would it be for a new programmer to start development on the project? How hard would it be to make large structural changes if the requirements change?

- System interconnections and dependencies should be minimized using design and tactics such as events (reduce code coupling)
- Clear and concise code
- Minimize unnecessarily performance-costly code (e.g. FindObjectsOfType in Update())
- Logical project file organization
- Efficient use of design patterns to improve code **legibility** and **scalability** of the game code
  Note: Organization marks may also be scaled according to the scale of the project. For example, an immaculately clean but near-featureless project will receive lower marks than a complete project with average organization

Flexibility of Criteria

- These criteria are designed to challenge you in specific ways to design scalable systems and use many features of the Unity Engine
- You may change aspects of the project to suit your particular vision, however you should discuss with your professor to ensure your vision still presents similar coding challenges and uses similar Unity features
- The game *does not have to visually or thematically resemble Pokemon at all.* Encounters do not need to be about physical combat. Hit points and damage are not required mechanics.
- The below breakdown is a *guide*, not the law. Be creative!

Marking Guide

Task	Weight	Description
Source Code Submission - Mandatory	1	Commit history – 1 Free of large unnecessary files – 1
Itch.io Game Submission - Mandatory	6	Page details – 1 Developer Log – 5
Acknowledgements	1	Credits – 1

- Mandatory		
Menus & Options	3	Audio settings - 1 Title screen/Game Over/Restart - 1 Credits & Title screen - 1
Overworld	8	Well-designed – 4 Lighting – 2 Correct collision – 2
Gameplay Encounter System	14	Random encounters in world – 3 Ability-based conflict system – 5 Useful gameplay feedback – 5 Win/Loss – 1
Ability System	20	Character, UI, and Abilities all decoupled – 8 4+ abilities w/unique effects – 4 Gain abilities – 4 Flexible design – 4
Encounter Opponents	16	Uses decoupled abilities just like a player – 2 Multiple types – 6 Can make intelligent decisions with any ability set – 8
Game Theming, VFX/SFX	8	SFX/VFX - 4 Music system - 2 Asset coordination - 2
Saving/Loading	8	Save/Load position– 1 Save/Load Abilities gained in play, HP, Items, & other progression - 7
Overall project and code organization	20	Organized project – 5 Overall decoupling and scalability – 10 Clear code & information flow – 5
Fun	35	Game theme/mechanics/design are creative and/or have essence of fun Game provides relevant information and meaningful decisions to players
Achievement System	10	Achievement system -Popups the moment an Achievement is unlocked, holds for some time -A menu that shows a list of all locked and unlocked Achievements -Interesting Achievements spanning multiple different parts of play – taking X steps, defeating X opponents, narrowly winning a battle, defeating an opponent with a specific ability, opening the Achievement Menu. -Achievement system should not be directly coupled with existing game code
Total:	150/100	

Penalties

- Instructor provided code is allowed as a base, however using external code could result in a mark of zero for the entire assignment. If you use instructor-provided code/assets, please note that you will not be graded on these parts. You may only receive grades on work you have done

- **ALL** plagiarism rules apply: you must cite your sources for ideas and for assets in the project. If you use an idea or asset that is not yours (e.g. Youtube tutorial) you must cite it, specifying specifically which parts of your project were influenced by what —even if it is a derivative work (i.e. not directly copy-pasted).

Keep in mind that the game is intended to give students the opportunity to practice coding and using certain Unity features. Therefore, you should ask the professor before using any external package which may replace or otherwise change your use one of the following Unity features:

- Sprites
- Collision
- Tile Maps
- Lights
- UI panels, text, and buttons
- UnityEvents
- ScriptableObjects
- Mecanim animation
- Coroutines
- Sound
- Particle Systems
- Saving
- C# Scripting
- GameObjects & Prefabs