

MAJOR COURSE OUTPUT

CCPROG3 AY 2023-24 T1

INTRODUCTION

A role-playing game (**RPG**) is a game genre in which players advance through a main story quest, and often multiple side quests, in order to gain in-game rewards such as items or experience [[SOURCE](#)].

First popularized by Dungeons & Dragons (**D&D**) through pen-and-paper gameplay, RPGs today now typically come in electronic form, with an earlier example being the original Pokémon games released for the Game Boy console in 1996 [[SOURCE](#)].

For this project, you are to recreate a highly simplified version of an RPG game, based on the original Pokémon games. In Pokémon, the goal of the game is to collect all creatures from a given list, which can be done through area exploration, trading, or evolution. This project will mainly focus on two of these mechanics, namely area exploration and evolution.

PROGRAM DESCRIPTION

The goal of the program is to catch and evolve creatures. The program starts by asking the user to select one (1) evolutionary level 1 (i.e. **EL1**) starter creature, which is automatically added to their inventory, and set as their active creature. Then, the user is shown the menu screen, which contains the following **OPTIONS** : [1] View Inventory, [2] Explore an Area, [3] Evolve Creature, [4] Exit.

OPTIONS [1], [2], and [3] are further discussed in the [INVENTORY](#), [AREAS](#), and [EVOLUTION](#) subsections respectively. Take note that **OPTION** [2] should first ask the user which specific area they wish to explore, before showing the actual area screen.

The program specifics are further discussed below. Sample screens for some features can be seen in [APPENDIX A](#).

CREATURES

The goal of the program is to catch and evolve creatures. Creatures have a name, type, family, and evolution level (**EL**). The list of creatures can be seen in [APPENDIX B](#).

INVENTORY

The inventory stores the user's list of captured creatures. Newly caught creatures are automatically added at the end of the list. Users can also change their active creature in the inventory. The active creature is the creature that attacks whenever the user selects the **ATTACK** option during battle (see [BATTLE PHASE](#)). Users can only have one (1) active creature at a time.

This screen should **SHOW** the following : [1] each creature's image^[★], name, type, family, and **EL**, [2] the active creature, and [3] an option to change the active creature.

[★] **NOTE** : Only applicable for **PHASE 2**.

AREAS

An area is where users can encounter creatures. There are three (3) areas in the program, and each area is represented by an **N x M** number of tiles. **AREA 1** is **5 x 1**, **AREA 2** is **3 x 3**, and **AREA 3** is **4 x 4**. The area screen must show each area accordingly (e.g. if an area's dimensions form a grid, then the program should show it as a grid).

The creatures encountered for each area also differ. **AREA 1** can only yield **EL1** creatures, **AREA 2** can yield both **EL1** and **EL2** creatures, and **AREA 3** yields **EL1**, **EL2**, and **EL3** creatures.

When entering an area, the user starts at tile [0, 0]. They can then move one tile **UP**, **DOWN**, **LEFT**, and **RIGHT**, as long as they remain within the bounds of the area. For example, in **AREA 1**, users can technically only move **LEFT** or **RIGHT**, since moving **UP** or **DOWN** will place them out-of-bounds. Make sure that the user's current location is obvious (e.g. there is a marker denoting their current position in the area screen).

Every time a user moves to a different tile, they have a **40%** chance to encounter a creature. Encountering a creature opens up the battle phase. Users should also be given an option to exit the area, which returns them to the menu screen.

BATTLE PHASE

NOTE : For brevity, this section uses "Enemy" to refer to the encountered creature.

During this phase, users can do at most three [3] actions, before the Enemy "*runs away*", which ends the battle phase and returns the user to the area screen. Take note that the Enemy does **NOT** perform any actions, aside from automatically "*running away*", whenever the user's actions are fully consumed.

An action is consumed when users do one of the following : [1] **ATTACK**, [2] **SWAP**, and [3] **CATCH**. Additionally, users can also “*run away*” from the battle, if they do not wish to catch the Enemy. All Enemies have a starting health of 50. Choosing the **ATTACK** option deducts the Enemy’s health by a damage value computed using the following **FORMULA** :

RAND(1, 10) * AC_EL	
RAND	A random value, inclusive of the given range.
AC_EL	The user’s active creature EL.

If the active creature’s type is stronger than the Enemy’s type, then the damage value is further multiplied by 1.5 (**FIRE** > **GRASS**, **GRASS** > **WATER**, and **WATER** > **FIRE**). Take note that if the Enemy’s health gets reduced to 0, it does **NOT** get caught, and the user is returned to the area screen.

Choosing the **SWAP** option allows the user to change their active creature with a different creature from their inventory. Do **NOT** allow the user to select this option if they only have one (1) creature in their inventory (which is already the currently active creature).

Choosing the **CATCH** option attempts to catch the Enemy. The catch rate (i.e. percentage of succeeding a catch) is based on the following **FORMULA** :

(40 + 50 - EN_HP)%	
EN_HP	The Enemy’s current health.

If the catch attempt was a **SUCCESS**, then the Enemy is added to the inventory, and the user is returned to the area screen. If the catch attempt **FAILED**, but the user still has remaining actions, then they may continue with the battle phase. If the user has **NO** actions left, then they are returned to the area screen.

This screen should **SHOW** the following : [1] the active creature’s name, type, and EL, [2] the Enemy’s health, image^[★], name, type, and EL, [3] the damage value (only after **ATTACK**), [4] the inventory (only during **SWAP**), and [5] the catch result (only after **CATCH**).

[★] **NOTE** : Only applicable for **PHASE 2**.

EVOLUTION

Evolution is done by selecting two (2) creatures from the user’s inventory in an attempt to generate an “*evolved*” creature. The evolution is a **SUCCESS** if the selected creatures are of the same EL and family. Else, the evolution **FAILS**.

Upon **SUCCESS**, the selected creatures are removed from the inventory, and the “*evolved*” creature is added at the end of the inventory. The new creature is the next **EL** of the same family. Creatures of **EL3** are **NOT** allowed to be selected for evolution.

This screen should **SHOW** the following : [1] each selected creature’s image^[★], name and **EL**, [2] evolution prompts (**SUCCESS** or **FAIL**), and [3] evolved creature (upon successful evolution).

[★] **NOTE** : Only applicable for **PHASE 2**.

SUBMISSION

All deliverables are to be submitted via Canvas. Submissions made in other venues will **NOT** be accepted. Please make sure to take note of the deadlines specified on Canvas. **NO** late submissions will be accepted.

PHASE 1

1. The program **FEATURES** include : [1] all **EL1** creatures, [2] the complete **INVENTORY** functionality, [2] the complete **AREAS** functionality, but only for **AREA 1. EVOLUTION** is **NOT** included.
2. Design and implementation should exhibit proper object-based concepts, like encapsulation and information hiding.
3. Text-based output simulation of the features.

PHASE 2

1. The program **FEATURES** include **ALL** features outlined in this document.
2. Design and implementation should exhibit proper object-oriented concepts, like inheritance, polymorphism, and method overloading/overriding.
3. GUI-based output simulation of the features.
4. The program should also utilize the MVC architecture.

DEMONSTRATION

The mode of delivery for the **PHASE 1**’s demo is left to the discretion of your instructor. If your instructor asked for a video demo, then please take note of the following :

However, **PHASE 2**’s demo is expected to be **conducted live** (whether F2F or as a synchronous Zoom meeting). During the demo, **ALL** members of the group must be present. Apart from the standard Q&A, a demo problem will be given to the group as part of the demo.

A student or group who is **NOT** present during the demo, or who fails to convincingly answer questions regarding the design and implementation of the submitted project, will incur a grade of **0** for that project phase.

During the demo, it is expected that the program can successfully compile and run. If the program does **NOT** run, the grade for that phase is **0**. However, a running program with complete features may **NOT** necessarily get full credit, as implementation (i.e. code) and design (i.e. UML) will also be checked.

COLLABORATION AND ACADEMIC HONESTY

This project is meant to be worked on as a pair (i.e. **MAX** of **2** members in a group). In exceptional cases, a student may be allowed by their instructor to work on the project alone, however, **PERMISSION should be sought** as collaboration is a key component of the learning experience. Under **NO** circumstance will a group be allowed to work on the **MCO** with more than **2** members.

A student is **NOT** allowed to discuss or ask about design or implementation with other persons, with the exception of the teacher and their groupmate. Copying other people's work and/or working in collaboration with other teams are **NOT** allowed, and are punishable by a grade of **0.0** for the entire **CCPROG3** course, and a case may be filed with the Discipline Office. In short, do **NOT** risk it, the consequences are **NOT** worth the reward.

BONUS

A **PHASE 2** submission can receive at most **10** bonus points based on additional features, if and only if the submission has met the basic requirements (i.e. exemplary for program correctness and design of classes and their relationships). There are **NO** bonus points for **PHASE 1**.

Awarded points may vary based on the complexity of the additional feature. Assets that make the program more engaging, or add to the overall user experience may also be awarded points. Groups have the freedom to add bonus features as they see fit. However, added features must **NOT** permanently alter the base requirements of the project. There should always be a quick and easy way to test the minimum requirements in the final submission, despite any added features. If a bonus feature needs to alter the base requirements, the group must provide a setting that will momentarily disable such alterations. Not being able to test the expected base

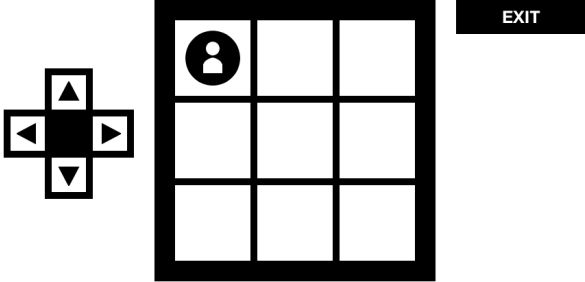
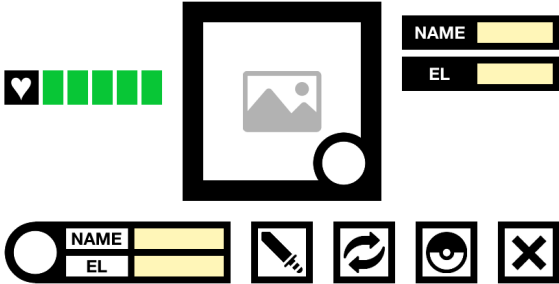
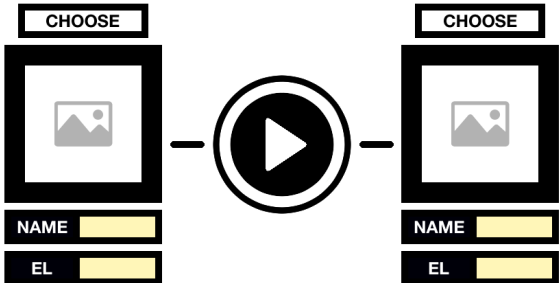
requirements will most likely result in deductions to the submission's score, as well as the bonus points not being counted.

To encourage the use of version control (such as Git), up to 4 bonus points may be awarded for proper Git usage. Although version control will **NOT** be taught in class, you are highly encouraged to utilize it to help with collaboration, and for accountability regarding member contributions. Awarded points may vary based on how the group was able to leverage the usage of version control (e.g. small and often commits, descriptive commit comments, contributions from all members, branching).





















Note that bonus points are capped to 10. For example, if a group was awarded 10 points for the implementation of additional features and 4 points for effective usage of version control, only 10 points will be awarded. Lastly, do **NOT** waste your effort on additional points when the project requirements have not been met!
















APPENDIX A SAMPLE SCREENS

The following images are screen mock-ups to help visualize the intended program. Your **PHASE 2** screens do **NOT** need to be exact replicas of the images below. However, take note that the user options (i.e. buttons) shown in each screen must be present in your final output.

 A screen mock-up for the 'AREA SCREEN'. On the left is a cross-shaped directional pad with arrows pointing up, down, left, and right. To its right is a 3x3 grid of squares. The top-left square contains a black circle with a white person icon. To the right of the grid is a black rectangular button labeled 'EXIT' in white capital letters.	<h3>AREA SCREEN</h3> <p>INFORMATION :</p> <ul style="list-style-type: none">■ Area tiles (based on proper dimensions) and user token. <p>BUTTONS :</p> <ul style="list-style-type: none">■ UP, DOWN, LEFT, RIGHT, and exit.
 A screen mock-up for the 'BATTLE SCREEN'. At the top left is a health bar consisting of a heart icon followed by five green rectangles. To its right is a square frame containing a placeholder image of a landscape and a small white circle at the bottom right. Further right are two input fields: 'NAME' and 'EL', each with a yellow rectangular box for text. Below these elements is a row of five buttons: a circular button with a white slider, a button with a pencil icon, a button with a circular arrow icon, a button with an eye icon, and a button with an 'X' icon.	<h3>BATTLE SCREEN</h3> <p>INFORMATION :</p> <ul style="list-style-type: none">■ Enemy's health, image, name, EL, and type.■ Active creature's name, EL, and type. <p>BUTTONS :</p> <ul style="list-style-type: none">■ Attack, swap, catch, and run.
 A screen mock-up for the 'EVOLUTION SCREEN'. It features two identical panels on either side of a central circular button with a white play icon. Each panel has a 'CHOOSE' label above a square frame containing a placeholder image. Below each frame are two input fields: 'NAME' and 'EL', each with a yellow rectangular box for text.	<h3>EVOLUTION SCREEN</h3> <p>INFORMATION :</p> <ul style="list-style-type: none">■ Selected creature ONE's image, name, and EL.■ Selected creature TWO's image, name, and EL. <p>BUTTONS :</p> <ul style="list-style-type: none">■ Choose creature ONE, choose creature TWO, and start evolution.

APPENDIX B CREATURE LIST

	TYPE	EVOLUTION LEVEL		
		LEVEL 1	LEVEL 2	LEVEL 3
A FAMILY				
		STRAWANDER	STRAWLEON	STRAWIZARD
B FAMILY				
		CHOCOWOOL	CHOCOFLUFF	CANDAROS
C FAMILY				
		PARFWIT	PARFURE	PARFELURE
D FAMILY				
		BROWNISAUR	CHOCOSAUR	FUDGASAUR
E FAMILY				
		FRUBAT	GOLBERRY	CROBERRY

	TYPE	EVOLUTION LEVEL		
		LEVEL 1	LEVEL 2	LEVEL 3
F FAMILY				
		MALTS	KIRLICAKE	VELVEVOIR
G FAMILY				
		SQUIRPIE	TARTORTLE	PIESTOISE
H FAMILY				
		CHOCOLITE	CHCOLISH	ICESUNDAE
I FAMILY				
		OSHACONE	DEWICE	SAMURCONE

APPENDIX C REFERENCES

All creature images are taken from the respective Pokémon pages of the PokéSweets wiki [COOKBOOK](#).