Minor Project: Rezyak

Contents

Overview

In this project, your group of two to three will:

- ☐ Design and implement the Rezyak card game.
- ☐ Keep track of your progress using version control.
- ☐ Use software engineering tools to help create quality code.

Minor Project: Rezyak

About the Game

Rezyak is a casino game played in the Rigel-7 system.

A. Components

1. The game uses two spinners that each produce random numbers from different ranges of numbers.
   a. One spinner produces random numbers from the 2-5 range.
   b. One spinner produces random numbers from the 0-7 range.
2. The currency used is the Rigel-7 Zephy[1].

B. Goal

1. Get as close to the limit of 17 without going over.

C. How to play

1. Place a bet according to the stakes you want to play:
   a. Low Stakes: 5 Zephy.
   b. Normal Stakes: 25 Zephy.
   c. High Stakes: 50 Zephy.
2. Choose a spinner.
3. If the player chooses not to spin, the game pays out according to the following chart:

Player's Total	Payout	If bet 20 Zephy, receives
< 11 or > 17	0%	0 Zephy
12	25%	5 Zephy
13	50%	10 Zephy
14	100%	20 Zephy
15	125%	25 Zephy
16	150%	30 Zephy
17	200%	40 Zephy

[1] 1 Zephy is equivalent to $6 from the Solarian country of Canada.

D. Setup

1. A player starts with 50 Zephy.
2. There are three types of spinners used, depending on what the player bets:
   a. Low Stakes: The spinner favours the player to get them to play more. This spinner produces more low numbers, which increases the likelihood of the player reaching the limit.
   b. Medium Stakes: The spinner is fair. All numbers are equally likely.
   c. High Stakes: The spinner cheats the player. This spinner produces more high numbers which increases the likelihood of the player going over the limit.

Instructions

A. Set up:

1. A project repository will be provided for your group.
   a. You will receive the URL by email after the group choice deadline.
   b. The repository contains a `PlaceboClass`, which you should delete.
2. Create a local clone of your assignment repository.
3. Run all tests from the command line to make sure you have started with a clean copy of the repository.

B. To complete the project:

1. Have a team meeting and determine who will be responsible for which tasks.
2. Create two different designs for the project.
   a. Determine the classes and methods your group will implement to write a program to let a person play Rezyak.
   b. At least one design must use inheritance and polymorphism.
3. Implement the design.
   a. The program is to run on the command-line and have a text-based interface.
      (a1) You are free to develop whatever text-based UI you choose. In other words, you decide the prompts and messages that the game will have.
      (a2) It is expected that the design will change during the implementation.
      (a3) You can use ASCII art.
      - The art must be found in one or more separate files that are read in to keep the code readable (this is also good SE practice).

- ♦ Games that use ASCII art will not receive a higher grade.
- ♦ Your time is better spent making sure that the program works instead of making it look great.

b. The project must be developed in C++.
   (b1) The classes you write must be saved in files with names in PascalCase. That is, the first letter of each word in the name, including the first one, must be capitalized (ie: MyGreatClass.h and MyGreatClass.cpp).
   (b2) All class header files must have the extension **`".hpp"`** and be saved in the include directory.
   (b3) All class implementation files must have the extension **`".cpp"`** and be saved in the src directory.
   (b4) The main function must be saved in a file named **`"main.cpp"`** and be saved in the src/project directory. You can alter the file that is already in that location.
c. Use branches on GitLab to reduce the frequency of merge conflicts. Do not delete the branches after merging them.
   (c1) The contents of the `main` branch will graded.
d. A `Makefile` is provided for running continuous integration. To get full marks, your software must:
   (d1) Compile on the department lab computers (`make project`)
   (d2) Pass style analysis (`make style`)
   (d3) Pass static analysis (`make static`)
   (d4) Pass memory leak checking (`make memcheck`)

Ideal regular work procedure and advice

A. Functional requirements:

1. Update your local version of the repository at the beginning of every work session by pulling from `origin`.
2. Always push your changes to the online repository at the end of every work session by pushing to `origin`.
3. Update the `README.md` regularly to keep track of who has completed what parts of the project.

B. Coding advice:

1. Create the skeleton:
   a. Create a minimal `main.cpp`. (Provided in this project)
   b. Create a header file and an implementation file with stub functions for all planned methods.
   c. Make sure the project skeleton compiles using `make project`.

   d. Make sure the project skeleton runs and does nothing.
2. Create a branch for each code implementation task.
   a. Implement classes that do not depend on unwritten classes first.
   b. Use main to test the methods as you can.
3. For each class or concrete class that inherits from an abstract class:
   a. Implement code in the following order:
      (a1) Constructors
      (a2) Mutator/setter methods used by the constructor, if any.
      (a3) Destructors
      (a4) Accessor/getter methods
      (a5) Mutator/setter methods
      (a6) Methods that do not depend on unimplemented methods.
   b. Check that the method you have written.
      (b1) Add a line to main that uses the method.
      (b2) Check all the tests.
         - The code passes the staticAnalysis test (`make static`).
         - The code passes the styleCheck (`make style`).
         - The code compiles using the Makefile (`make project`).
         - The new method runs correctly (`./project`).
         - The code has no memory leaks (`make memcheck`).

Design Deliverables checklist

Requirement 1: Reporting

- ☐ Indicate in the README.md file of the team's repository the members of the group and how they contributed to the design.
  - ○ This provides an opportunity to indicate to the instructors any team members who "have gone rogue," such as ghosting the team or trying to complete the project on their own.

Requirement 2: Content

- ☐ Submit the two designs to the Moodle workshop on the course's Moodle page.

- The design clearly shows:
  (1) The name of the class
  (2) The class attributes.
  (3) The class methods with return types and parameter types.
- The design is in one of two forms:
  (1) UML class diagrams in JPG or PNG format.
    (a) Must use the correct notation for inheritance.
    (b) Must use the correct notation for visibility.
    (c) Try to use the correct notation for dependencies, composition and aggregation.
  (2) A text document in PDF format.
    (a) Class names are sections with subsections for attributes and methods.
    (b) Visibility are subsections within the subsection.
    (c) Attributes and methods are given as a bulleted list.
- Each team member must submit a copy of the team's design.
  (1) Failure to submit your team's design on Moodle will result in you losing marks for the project, even if you contributed to the design. This is due to how the Moodle Workshop works.

Implementation Deliverables checklist

Requirement 3: Set up

- ☐ The URL of your GitLab repository is the team URL you were given for this project.
- ☐ The course instructor, lab instructor, and marker (username: mark2720) have been invited to your repository as reporters.

Requirement 4: Reporting

- ☐ Indicate in the README.md file the members of the group and the tasks that they completed.
  - This provides an opportunity to inform the instructors of any team members who "have gone rogue," such as ghosting the team or trying to complete the project on their own.

Requirement 5: Content

- ☐ A UML diagram showing the final design of the project.
  - Diagram shows:

(1) Class names
(2) Method names with their parameter types (don't need parameter names)
(3) Variable names
(4) Relationships (inheritance, dependency, composition)

- ☐ The required classes are implemented and used.
- ☐ All stages of the pipeline pass.
  - staticAnalysis (`make static`)
  - styleCheck (`make style`)
  - buildTest (`make project`)
- ☐ At the command line the program runs without fault
  - Program plays the game (`./project`)
  - There are no memory leaks (`make memcheck`).
- ☐ Evidence of using version control to manage group work.
  - Division of tasks is shown using branches with meaningful names.
  - Show progress in completing the project over the duration of the project by having at least ten commits on at least six different dates.
  - Stages of completion are documented by the use of meaningful commit messages.
    (1) Good commit messages tell the "what" and "why" for the commit (e.g. "Fixed style errors to pass style checking.")
  - Show that all members of the group contributed by having commits completed by each member of the group.

Requirement 6: Other details

- ☐ You have not altered any files that you shouldn't change.
  - The only files you may alter or add include:
    (1) The header files in include.
    (2) The implementation files in src and src/project.
    (3) .gitignore
    (4) README.md
- ☐ No unnecessary files have been committed to the repository.
  - None of the material that can be generated such as the documentation, coverage report, or executables.

- None of the directories or files used by IDEs, or your platform.