Final Project

CS 211

Winter 2022

Proposal due:	Friday, Feb 25 at 11:59 PM Central Time	(via Proposal survey)
Specification due:	Tuesday, March 01 at 11:59 PM Central Time	(via Specification survey)
Code due:	Friday, March 11 at 11:59 PM Central Time	(via Gradescope)
Evaluation Guide due:	Sunday, March 13 at 11:59 PM Central Time	(via survey distributed later)
Partners:	Yes; mark you partner on Gradescope on each submission	

Contents

1 Purpose

The goal of this assignment is to let you apply the programming skills you've acquired in service of your own creativity.

2 Getting it

While there isn't significant starter code for the final project, we've prepared a project ZIP file containing all the dependencies and CMake configuration you need to get started.

3 Project requirements

For the final project, you must implement a game (or other interactive, graphical program) using C++ and GE211. There are four steps to the project: proposal, specification, delivery, and evaluation:

Proposal You will submit the name and a brief synopsis of your game, with sufficient detail for us to have an idea what you would like to do. We may approve your idea, or we may ask you to alter it and resubmit.

Specification You will write a list of *functional requirements* (as discussed below) for your game. This is the set of features that you're planning to implement. As with the proposal, we may accept this or ask you to revise. Once approved, the functional requirements become the rubric for grading your implementation.

Implementation Then you actually implement and deliver the code of your program.

Evaluation Finally, you need to supply us with an "evaluation guide," as detailed below, that we will use to evaluate your program.

The game is expected to be of moderate complexity, perhaps 1.5 times as complex in terms of requirements as Homework 5's Brick Out or Homework 6's Reversi. We will be more precise about assessing this aspect of your proposal below.

4 Proposal

For your proposal, we just want to know what game you would like to make so that we can ensure that the difficulty level is reasonable (in both directions) before you get to work.

4.1 How to submit it

Please submit your proposal by filling out the proposal survey. We will review proposal and respond on a rolling basis, so the sooner you submit, the sooner you will get feedback that enables you to move on to the next step, specification.

4.2 What should go in it

If you are cloning an existing game (a good idea, since game design itself is really difficult!), you just need to tell us about that game. If it's old and sufficiently well known then the name could suffice. Here are examples of two proposals in that form:

> Ms. Pac Man

> I want to make Galaga.

If you are cloning a game that is newer or more obscure, it might help to include a link to a description, or you can describe it yourself. For example:

> I'd like to make a version of the game "Underwater Basket Weaving 6." For a good description of the game, see this Wikipedia article.

Finally, if you want to invent your own game, you may need to write your own description to convey the idea to us. For example:

> I would like to make a game that I am calling "Brick Out." It will have three elements: 1) a stationary array of rectangular bricks at the top of the screen, 2) a rectangular paddle at the bottom that moves horizontally and is controlled by the user, and 3) a circular ball that bounces in between, destroying any bricks it collides with. The player's goal is to destroy the bricks without allowing the ball to reach the bottom of the screen.

Images and diagrams are highly appreciated. Please try not to write more than 100–200 words.

4.3 What is not allowed

We will approve proposals as they come in and reject games that seem too complex or too simple. Often, modifications can be made to make the game more reasonably scoped for a class project, and we'll provide that feedback.

Some games are too simple to make for a good class project, or have demo code that we provide to you. Proposals for the following will not be accepted:

- Pong
- Snakes
- Space Invaders
- Flappy Bird
- Keyracer
- Bejeweled
- Asteroids

5 Specification

This is a list of 10–12 *functional requirements*—specific, identifiable things that your program will do. These features must be observable to a player, since we will play your game and use these requirements as a checklist for grading. (It's okay if some requirements are difficult for a player to reach, but you will have to later justify that you achieve those by reference to your code.)

5.1 How to submit it

Please submit your specification by filling out the specification survey.

We will review specifications and respond on a rolling basis, so the sooner you submit, the sooner you will get feedback.

After receiving feedback, you may not change your functional requirements without approval. They will be used for grading as specified below.

5.2 What should go in it

It may be a bit tricky to figure out the best granularity for describing functional requirements. It would not be good, for example, to have two separate requirements: "Pressing the left arrow key moves the player to the left," and "Pressing the right arrow key moves the player to right right." Instead, that should be a single requirement, perhaps: "The player is controlled by the arrow keys." This is a matter of taste and judgment, so see the example below for guidance, and then consult with the course staff or ask on Campuswire about how to specify your particulars.

As an example, here is a specification for Brick Out with nine functional requirements:

1. The bricks are initially placed in a grid at the top of the screen.
2. The paddle's x coordinate follows the mouse, while its y coordinate is fixed near the bottom of the screen.
3. In the dead state (the ball's initial state) the ball sticks to the paddle.
4. The player can release the ball, transitioning it from dead to live state, by pressing the space bar or clicking the mouse.
5. When the ball is released, it travels upward from the paddle with some initial velocity.
6. If the ball strikes the top or side of the screen, it bounces off.
7. If the ball strikes a brick, it destroys the brick and bounces off with a small, random boost to its velocity (TBD).
8. If the ball strikes the paddle, it bounces off like normal.
9. If the ball reaches the bottom of the screen, it transitions back to the dead state (and nothing else changes).

6 Implementation & delivery

Write your program. Have fun!

6.1 How to submit it

Homework submission and grading will use Gradescope. You must include any files that you create or change. If you use resource files, such as images or music, upload those, as well as your updated CMakeLists.txt. [1]

Per the syllabus, if you engaged in arms-length collaboration on this assignment, you must cite your sources. You may write citations either in comments on the relevant code, or in a file named README.txt that you submit along with your code. See the syllabus for definitions and other details.

Submit your files by uploading them directly to the Gradescope website. When you click on the assignment in Gradescope for the first time, you will get a window where you can upload files. You can drag-and-drop files or browse to select them. Make sure you include all the necessary files in the src/ and test/ directories! Submitting extra files is fine. To submit additional times, select the "Resubmit" button on the bottom right.

6.2 Partners

If you work with a partner, then you MUST register your partnership on Gradescope. From the Gradescope course page, click on the assignment, then at the bottom click the button labeled "Group Members". A dropdown menu will allow you to select the name of your partner, then click save to send them an email. Group members can also be removed by the same process. For more details see the help page on Gradescope.

Please remember to mark your partner on each submission.

7 Evaluation guide

Your proposal is worth 5% of your project grade, the specification is worth 10%, and the final code delivery is worth the other 85%. That 85% is further broken down into three components:

style	10%
model tests	20%
functional requirements	70%

We will assess style on our own, but for the other two points, we will need your help in the form of the "evaluation guide" described here.

7.1 How to submit it

Evaluation guide questions will appear on Gradescope as the self evaluation. So you should submit

[1] When reconstructing your project for grading, Gradescope puts source files whose names begin or end with "test" in the test/ directory, other source files in the src/ directory, and files with types it doesn't recognize in the Resources/ directory. So make sure you name any files that need to be in the test/ directory appropriately.

it by performing self evaluate. However, you should read this now so that you know what will be expected.

7.2 What will go in it

The evaluation guide will contain the following two components.

Favorite model tests (20%). We would like to see five *significant* model tests. Choose tests that you think best characterize your design and demonstrate how your model works. For each, provide a very short description of what the test is about, along with a line tag.

Functional requirement hints (70%). For the core of the evaluation, we will attempt to verify that your program meets the functional requirements from your specification. (This is why you need our approval for any changes to those requirements.) For each requirement, there are three ways that we will attempt this verification:

1. By playing the game and observing the requirement, for full credit.
2. By reading a model test that demonstrates that the game meets the requirement, for full credit. (You are free to reuse a favorite test here.)
3. By looking at the code that implements the requirement, **for 80% credit.**

You must provide a numbered list matching your list of proposed and accepted functional requirements from your specification. For each requirement, specify how we should attempt to check it:

1. For validation by playing, you need to provide instructions for how to play the game to a state where the requirement can be observed. If your game has multiple levels, difficulties, or modes, you may find it useful for your `main()` function to take an optional command-line argument to allow us to jump to a particular level. Also, if you believe there's a chance that we will have trouble validating a particular requirement by playing, you may also provide a test or code reference (options 2 and 3) as backup.
2. For validation by test, you need to provide line numbers for the relevant test or tests, along with sufficient explanation for us to understand why the test you tagged is evidence that the functional requirement in question is met.
3. For validation by implementation—the least preferred method—you need to provide line numbers for the relevant implementation code, along with sufficient explanation for us to understand why the code you tagged is evidence that the functional requirement in question is met.