Introduction

Welcome to CS1950n! In this first assignment, you will create the basic framework and UI system for your game engine, which you will continually develop for the rest of the semester. Additionally, you will create a simple Tic-Tac-Toe game on top of your game engine. All assignments will have three sets of requirements: a design check, primary requirements, and secondary requirements. You should complete them in that order. Completing a set of requirements awards you one point, except for primary requirements, which are worth two points. Remember to keep your engine code separate from your game code.

This project is probably going to be one of the biggest projects that you’ll code completely on your own, so it’s really important to pace yourself. Tic will be a HUGE amount of code, all of which will be foundational for the subsequent projects, but don’t let that intimidate you!

You’ve got this :)

Stencil Code

To get the stencil code for this assignment (and all future assignments), click on this GitHub Classroom invite link. Create a GitHub account if you don’t have one yet, and make sure Git is installed on your computer. If you’re not familiar with Git at this point, we’d recommend reading up on the basics here.

From there, it will create a new repository for you with the stencil code. In the top-right, there will be a dropdown with an HTTPS clone link. On your local computer’s console, navigate to the directory where you want this project to live, and enter git clone <link>. That should copy everything over.
Next, open IntelliJ and select “Import Project”, then select the root directory of the stencil code. If you don’t have IntelliJ set up yet, you can follow this guide to do so.

Here are some important classes to familiarize yourself with:

**Application** - This is the base class all your projects will start with. An example of its usage is already included in tic/Main.java. The barebones of this class has been written for you, as it contains all of the user input, tick, and draw methods. It’s up to you to fill it out. (Hint: you’ll probably want it to hold Screens.)

**Vec2d** - A class representing a two-dimensional vector of doubles. It comes with many of the basic vector operations (i.e. dot product, scaling, etc). You should use it for any numbers that come in pairs, such as size or position.

We’ve already made a tic and engine directory for you, with some starter code inside. (Don’t worry about the third debugger directory for now.) Feel free to add more as you see fit—it’s always better to be more organized than have a ton of files in a given package. You’ll be building off of this repository for the rest of the semester, adding more directories for projects and engine components as you go.

**Collaboration Policy**
Before you proceed, make sure to review our Collaboration Policy and sign the linked acknowledgement form by Friday, September 11. We won’t be able to grade your assignments until you do this!

**Design Check Questions (remember to sign up!)**

- How will you define an application?
- How will you define a screen?
- What plans for a UI kit do you have (bounding boxes, aspect ratios)?

Also, feel free to use this time to ask your TA any other questions you may have!

*Note:* if there are no slots left or you can't make any of the times, email us!
Primary Requirements

Primary Engine Requirements

- Your handin must meet all global requirements.
- Your handin only crashes under exceptional circumstances (edge cases).
- Your engine must separate an application from screen and support an application with multiple screens.
- Your engine must draw the current screen on every “draw” event (originating from support code). Each screen must be able to define the way in which it is drawn independently from other screens.
- Your engine must be able to process mouse events (originating from support code) and allow each screen to define how they are handled.
- Your engine must be able to process keyboard events (originating from the support code) and allow each screen to define how they are handled.
- Your current screen must update itself on every “tick” event (originating from the support code).
- Your engine must be able to process resizable windows (originating from support code) and adjust the internal state when the draw area is resized. It must also ensure that size information is preserved when the current screen changes – if the current screen changes after a resize event, the new current screen must be aware of the correct window size as well.
- Your engine should have a basic UI toolkit. At a minimum, this toolkit should allow a game to display text and rectangles.
- Your engine must have a correct and easily extensible implementation of a button.

Primary Game Requirements

- Your handin must meet all playtesting requirements.
- A 3x3 square board must be accessible from the screen once the application is run (either directly or through a menu).
- An X or an O must appear on a box when that box is clicked
- Your game must implement the rules of Tic-Tac-Toe: two players, X and O, take turns marking squares on a 3x3 grid with their respective symbols. If a player succeeds in placing three symbols in a horizontal, vertical, or diagonal row, that player wins. If all the squares are filled without either player completing a row, the game is a draw. Both players can be human players: you do not need an AI opponent for this assignment.
- Have at least two screens: an in-game screen and another screen, such as a title screen.
- Clearly display which player’s turn it is.
- At the end of the game, effectively communicate which player won, or if it was a draw.
- Display the state of the game on a square board that scales with window size. The board must remain square at all times, no matter the window's aspect ratio.
- Your game must implement keyboard events (e.g., exit game on escape).
- Your game must never crash.

Secondary Requirements

Secondary Engine Requirements
- Your engine must meet all primary engine requirements.
- Your buttons should display differently when they are hovered.
- Each player’s turn should have a time limit. If the player does not make a move when the time expires, it becomes the other player’s turn and the timer resets.
- Show the countdown timer (e.g., shrinking bar, text in seconds).
- It must be possible to start a new game without restarting the program.

Tips

Drawing UI Elements

For your engine, you’re probably going to want to allow your user to color, style, rotate, etc, each UI element individually. The issue is that the `GraphicsContext` doesn’t know about the different objects that it is drawing, it just knows how to draw basic shapes based on its current set of state variables (i.e. fill, stroke, etc). So, setting one of these variables to draw one element will also result in the next element being drawn the same way. One way of approaching this problem is to cache the state of the `GraphicsContext` between drawing different UI elements. You can do this easily by using the native `save` and `restore` methods. [Here](#) are the Javadocs that explain it further.

Resizing
When resizing your UI elements, you might find it helpful to take advantage of the `setTransform` method of the `GraphicsContext`. Just be aware that you are only transforming the drawing space so you'll have to ensure that you take this into account when dealing with the location of mouse events. This is one way of approaching it; you can also consider just passing down a scale factor down to your UI elements.

**Suggested Extras**

If you meet the requirements listed above, you will receive credit for the assignment. If you have extra time and wish to make your game more interesting, you might want to implement some of these extra features. Note that you cannot receive extra credit for going beyond the assignment’s requirements, but the students who watch your demos will certainly appreciate it if you make it more fun to play.

- Have UI elements that fill as much space as they can while keeping the board square, instead of drawing UI elements at a fixed size.
- Let the user pick a larger-size board, such as 4x4, 5x4, or even NxN.
- Add a simple AI player and the option to play against the computer. It could even support multiple difficulties if you do this, remember that the option to play with two humans must still work as in the original requirements.

**Demo Video**

Once you’re done, record a short demo video of your project and upload it to the #demos Slack channel. You can either record your screen or just film it with a phone; it's just an informal way to stay up to date with what your classmates are working on. We highly encourage you to check out what your classmates have made and leave any comments you might have!

Along with the video, we ask that you provide some sort of commentary on what you worked on: what you’re happy about, what you found difficult, or so forth. You can just write a few sentences down and send it alongside your video.

**Handing In**

1. Push your final handin commit
○ Fill in your README template, as well as the INSTRUCTIONS file.
○ Run `git add .` and `git commit -m "<message here>"`
○ Verify you’ve committed all your changes with `git status`.
○ Push your code with `git push`. Check your repository online to verify that everything is there.

2. Create a release for this handin
   ○ On your repository page, click on “Create a new release” on the right-hand sidebar
   ○ Format your tag name as **Tic1.0**
     ■ If you want to handin a later version, just increment the version number and we’ll take the latest one
   ○ Set the target to the commit you just pushed (do not hand in the master branch)

   ![Git Push](img)

   ○ Fill in the rest of the form accordingly and publish your release.

3. Join our Gradescope course using this entry code: **M6B4BR**.
   ○ When prompted to fill in your name, fill in your **Banner ID**. This part is very critical: it’s how we link your GitHub Classroom repository with Gradescope.
   ○ You never need to turn in anything on Gradescope; the TAs will set up that part for you. Simply releasing your commit on GitHub is enough.