Project 2: MildaFlix✨

Support

In addition to the Design Check and the follow-up personal hours, you can feel free to come to TA Hours (http://cs.brown.edu/courses/csci0111/spring2022/calendar.html), or post on Ed (https://edstem.org/us/courses/18932/discussion/) for help with the project.

You got this! :)

Deadlines

Project 2 Partner Form Deadline: Wednesday March 23, 11:59PM EST

- Please have **one member** of your group submit the partner form. **You may not work with your Project 1 partner.** If you do not submit the form, you **will not receive a design check**! If you can't find a partner, fill out the form anyways! We will randomly match you with someone else! Once you have a partner, you will be emailed by a TA on Friday, March 25 to sign up for a design check appointment.

Design Checks: Tuesday, April 5 to Thursday, April 7

- One group member must submit the design check to Gradescope at least 12 hours before meeting with your design check TA. Ensure both partners are added to the submission

Final Handin Deadline: Thursday April 14, 11:59PM EST

Movie Pyreting

It's 2008. Milda is watching the cinematic masterpiece Twilight at the Providence Place Mall for the fifteenth time. The film has completely altered her views on life, time and love. In her inspired state, she has a vision – everyone in the world *must* see Twilight.

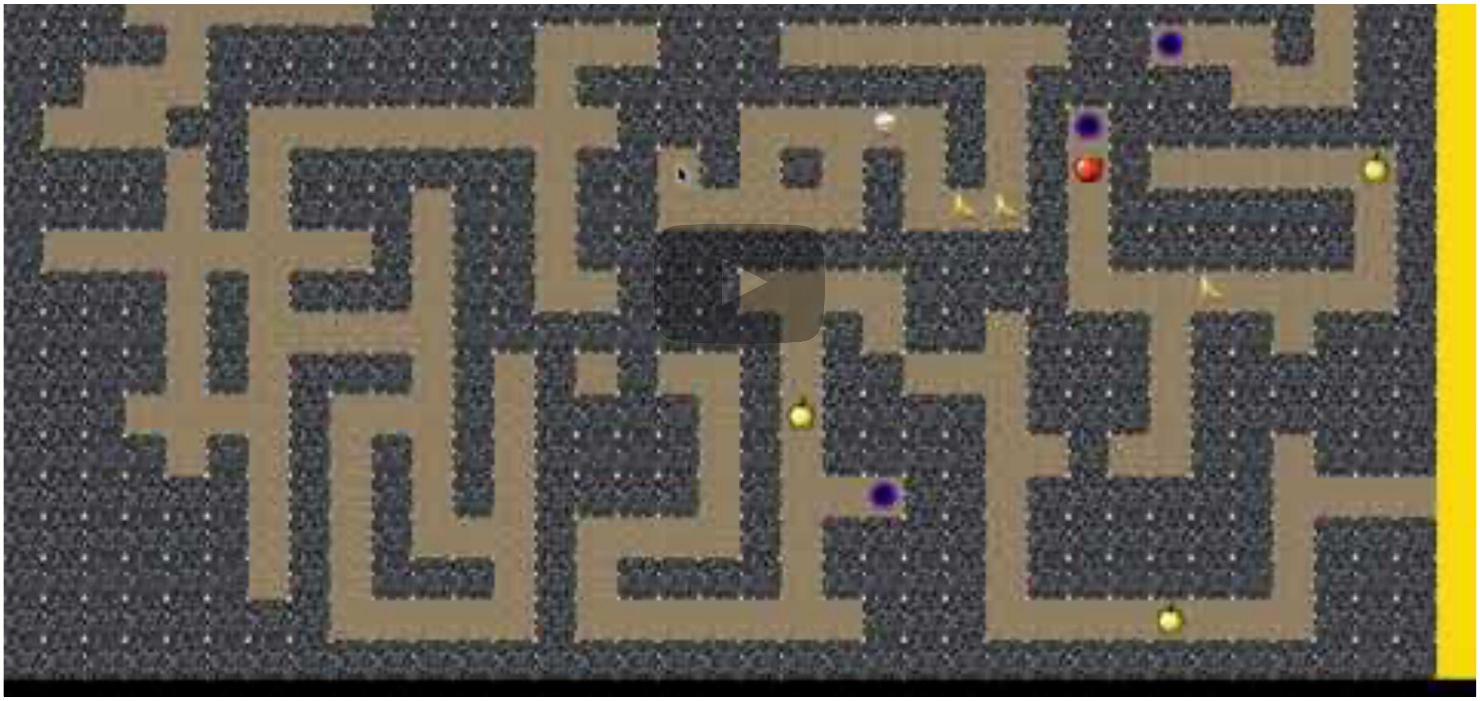
She steps into Showcase Cinema with a camera and a dream, ready to pirate Twilight for the masses.

...

Bella and Edward dance. Milda's in tears – again.

Hey! The movie theater person catches her recording. She needs to get out of there quick, but there are many obstacles in her way. Can you help Milda make it out of the theater and upload Twilight?

Summary



You will be creating a maze game. The user will navigate Milda around the maze using key presses and mouse clicks. The maze will contain some special items (gadgets), such as tickets to completely restore Milda's strength (shown here as the yellow bar on the right), popcorn to give them a small boost, wet floor signs to deplete their strength, or wormholes to allow Milda to teleport. The goal: help Milda safely make it back to her computer at the end of the maze.

Note: the video above shows the maze functionality with different special items. For MildaFlix, the item correspondence is: banana → popcorn, apple → tickets, tomato → wet floor sign, blue circle → wormhole, and exit → computer

The maze will be populated based on a configuration Google Sheet which you can modify to make your own maze. The game will be played using a Pyret `reactor`. The reactor will run on a `GameState` datatype that you define; the `GameState` data will evolve as the game progresses.

For the design check, you will be laying out the datatypes that will capture the necessary game information, and thinking through the tasks required to make the game run. **A lot of the work for this project will happen for the design check (you will**

need to write and submit some code).

For the final handin, you will be implementing the functions and reactor required to make the game run.

After all of the design checks are done, we will provide our solution to the design check phase, which you can use if you wish in producing your implementation for the final handin.

Project Learning Goals

- Creating and using your own datatypes.
- Using recursion to process various types of lists.
- Breaking down complex goals into discrete, solvable tasks.
- Using tables for code configuration rather than as datasets.

The Game

Maze

You will make a maze. The maze we provide in the Google Sheet is 35 squares per "row" and 19 squares per "column" (19×35), but you can add or remove rows, columns, and items as you please and make the maze your own! The goal of the game is for the (human) player to navigate Milda to the computer at the exit.

The maze will also have some number of special items (gadgets or portals) scattered throughout that Milda can collect and use. The items will be populated based on another Google Sheet.

The entire maze should be surrounded by one layer of walls (except at the end location), so you don't have to deal with the player moving outside of the maze and it is clear to the player where they need to navigate to in order to complete the game.

For the project, you will make a) the code to "render" (draw) the maze, its items, and Milda, and b) the logic required to allow a (human) player to use the mouse and keyboard to navigate the player around the maze.

The Player and Character

The current position of the player in the maze is represented with an image of Milda. The player moves Milda using the `W` (up), `A` (left), `S` (down), and `D` (right) keys. The player cannot walk through walls of the maze, but can move through empty space. For the Milda image, choose from the options in this directory (https://cs.brown.edu/courses/csci0111/spring2022/assets/data/project-2/) (see Implementation Details for more).

As an **optional** add on, you have the choice to make the character turn based on the direction they are walking (as demonstrated in the video at the top of the assignment). Implementing this feature will give you extra practice with datatypes and code design (but again it is entirely optional, and you will not receive extra credit).

Items

You will implement either gadgets or portals into the game. **You only need to implement one of these to get full credit**. You may do both if you want, but you will not get extra credit. However, if you do both, it will be fun and we will grade the two implementations individually–-the grade that is higher will be your grade for the project.

There can be an arbitrary number of these items placed on the maze. Items are picked up when the player moves into their cells; gadgets are immediately consumed, and portals are "held on to" until use. If multiple portals are collected, Milda will be able to "hold on to" all of them - and will only lose one each time they use a portal.

Either option will be difficult in its own way; we do not think either is significantly easier than the other.

- **Gadgets**: If you implement gadgets, your character needs to have *stamina*. Stamina is a measure of energy often used in video games. In the MildaFlix, your character's stamina depletes by moving. If your character runs out of stamina, it's game over.

  There are three things you can encounter in the maze: tickets, popcorn and wet floor signs.

  - Finding **tickets** 🎟️ replenishes Milda's stamina to its original value.
  - Finding a **popcorn** 🍿 heals Milda for some fixed amount of stamina.
  - Stepping on a **wet floor sign** ⚠️ reduces Milda's stamina to some low, fixed amount.

  For example, if the character's stamina is 20 and they move onto a wet floor sign, they may move down to 7 stamina. Then moving onto a popcorn may heal them to 7 + 8 = 15 stamina, and finding tickets will heal them to their original stamina (say, 30). Normal movement reduces stamina by 1 per cell.

  There should be a visual indication of the character's stamina (in our sample, it's the yellow bar on the right, but you can handle this differently if you wish).

- **Portals**: The portals in this maze will be represented by the **wormhole** 🌀 image.

  Your character can carry one or more portals (you may choose the limit), but starts with no portals. When the character has a portal and the user clicks some square on the screen, the character will move there *as long as the cell is within some reasonable range of the character's current location.* You can choose this range, but make sure it doesn't allow the user to easily teleport to the end of the maze!

  Each portal can be used only once. Your game screen should have some visual indication of how many portals the player has (simply a number using the Image library's `text` (https://www.pyret.org/docs/latest/image.html#%28part._image_text%29) function is fine).

To compute how far the user is trying to move the character, you can use the distance formula, with the change in the x-coordinate Δx and the change in the y-coordinate Δy:

$$D = \sqrt{\Delta x^2 + \Delta y^2}$$

For example, if the player is at $(3, 8)$ and wants to move to $(6, 4)$, $D = \sqrt{(6-3)^2 + (4-8)^2} = \sqrt{9+16} = 5$. This is a "birds-eye view" of distance.

In your game, if a player with a portal clicks on a cell, and the distance between the current and clicked cells is within the threshold you choose, the character moves to the new cell and uses up a portal. If the distance is larger than the threshold, nothing changes (the character stays in the same place and keeps the portal for another try).

Starting, Winning, and Losing

The player/character will start at a location of your choosing. The information of where they start should be hardcoded through constants in your code.

The player wins the game upon reaching the **computer** 💻 at an exit location that you set. This information should also be hardcoded through constants. We recommend having the end location on one of the edges of the maze so that it is clear when playing the game where the exit is. If your exit is not on the edge of the maze, you'll have find some other way to make it clear to the player where they have to go.

The player loses the game if they run out of stamina. If you don't implement gadgets, the player doesn't have to worry about losing.

Upon winning or losing, the game can simply close. You may also choose to somehow inform the player that they have won or lost (as was done at the end of Lab 2), but this is not a requirement.

> A video-based overview of the spreadsheets and how the gameplay works are in this video (https://www.youtube.com/watch?v=Zy3hWHe4X20&feature=youtu.be).

Implementation Details

Configuring the Maze (in Google Sheets)

The game will be populated based on your copy of a spreadsheet:

- Gadgets spreadsheet: link (https://docs.google.com/spreadsheets/d/1VI21BIZMZGfxU43gf1tnd8VWimNioKH5iVQLTR6ZaLs/edit?usp=sharing)
- Portals spreadsheet: link (https://docs.google.com/spreadsheets/d/1s5DpnEhJsiXYHKDUJvbi4unTJKCFb0dog9IZPhwb4mU/edit?usp=sharing)
- Both items spreadsheet: link (https://docs.google.com/spreadsheets/d/1Nt2JKwBK8gtVOQ7f5UtU-2K-P9WTLb7g3S6upqLK2cg/edit?usp=sharing)

Use the spreadsheet appropriate for your item implementation. Feel free to modify the maze layouts and item populations! See #2 in the "Support Code" section (below) for how to load the sheets.

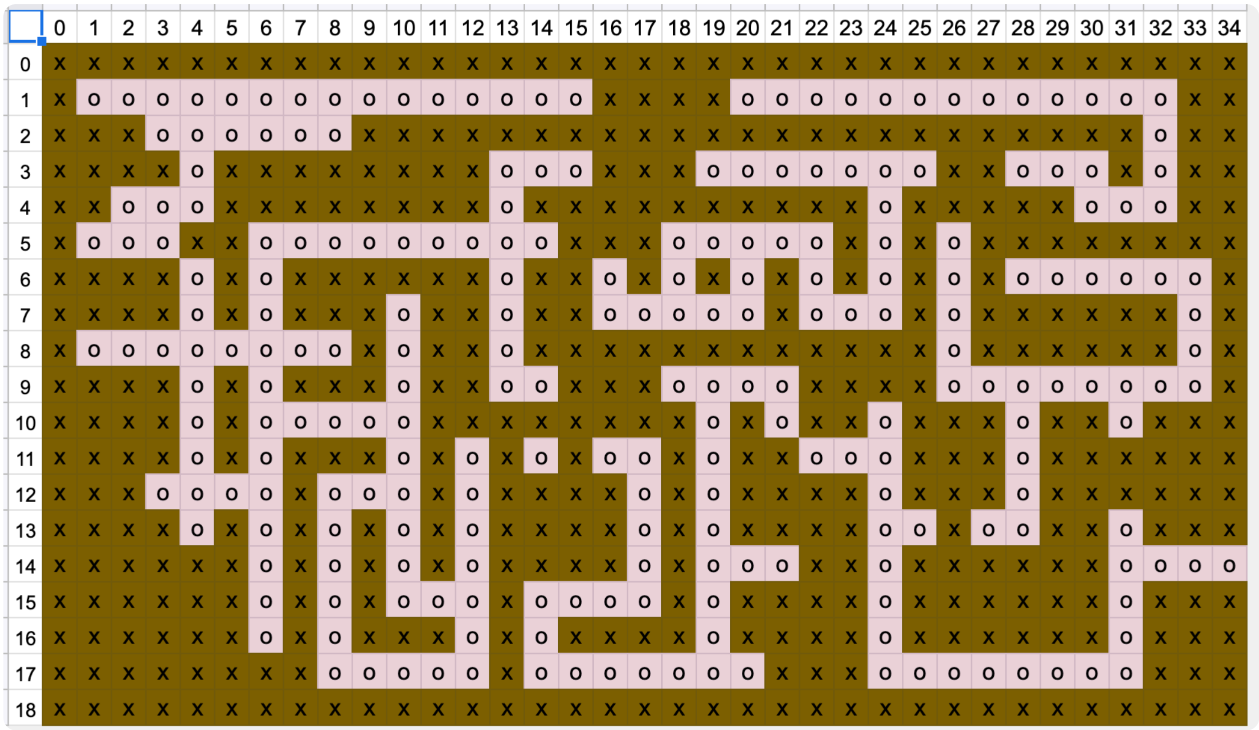
Each spreadsheet has two sheets: `maze-layout` and `items`.

- The `maze-layout` sheet determines which parts of the maze are walls and which are blank and therefore where the player can walk. "x" corresponds to a wall, and "o" corresponds to a floor (tile).

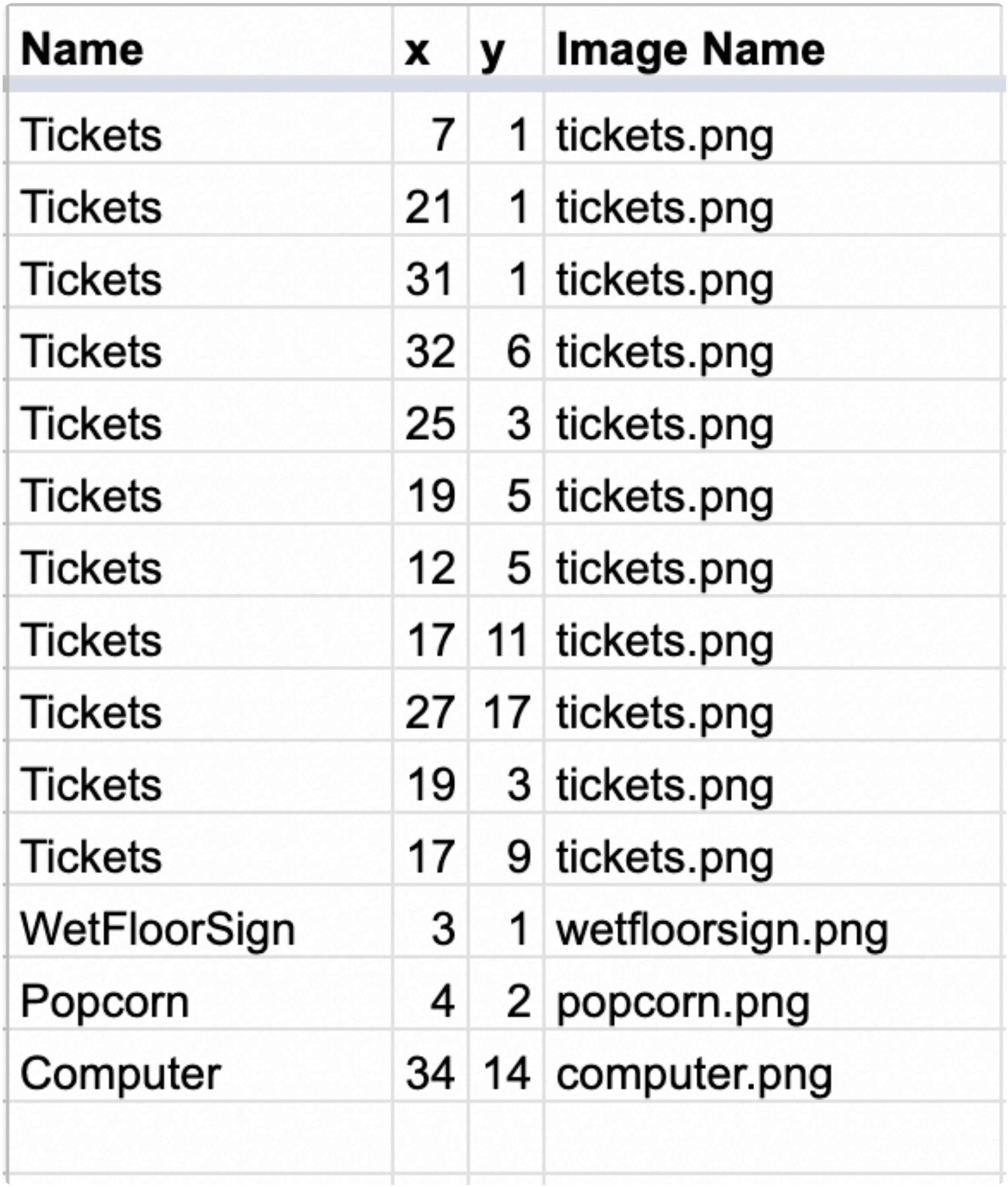
- The `items` sheet contains a list of items that will be placed on the maze. The x column is the distance from the left side of the screen, and the y column is the distance from the top of the screen. You will need to edit this sheet to put items in the appropriate places. Check out the images in this folder: link (https://cs.brown.edu/courses/csci0111/spring2022/assets/data/project-2/index.html)

> **Note 1:** Positions are zero-indexed; the row `"Tickets", 1, 7, "tickets.png"` corresponds to an 🎟 being placed in the 2nd column and the 8th row.
>
> **Note 2:** You do not need to do any error checking for items being placed in invalid locations (like on walls or off the maze).

> Do not adjust the sizes of the images that we provide. You should not rescale images.

Name	Image
Maze Sheet	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
0	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
1	x	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	x	x	x	x	o	o	o	o	o	o	o	o	o	o	o	o	o	x	x
2	x	x	x	o	o	o	o	o	o	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	o	x	x
3	x	x	x	x	o	x	x	x	x	x	x	x	x	o	o	o	x	x	x	o	o	o	o	o	o	o	x	x	o	o	o	x	o	x	x
4	x	x	o	o	o	x	x	x	x	x	x	x	x	o	x	x	x	x	x	x	x	x	x	x	o	x	x	x	x	x	o	o	o	x	x
5	x	o	o	o	x	x	o	o	o	o	o	o	o	o	o	x	x	x	o	o	o	o	o	x	o	x	o	x	x	x	x	x	x	x	x
6	x	x	x	x	o	x	o	x	x	x	x	x	x	o	x	x	o	x	o	x	o	x	o	x	o	x	o	x	o	o	o	o	o	o	x
7	x	x	x	x	o	x	o	x	x	x	o	x	x	o	x	x	o	o	o	o	o	x	o	o	o	x	o	x	x	x	x	x	x	o	x
8	x	o	o	o	o	o	o	o	o	x	o	x	x	o	x	x	x	x	x	x	x	x	x	x	x	x	o	x	x	x	x	x	x	o	x
9	x	x	x	x	o	x	o	x	x	x	o	x	x	o	o	x	x	x	o	o	o	o	x	x	x	x	o	o	o	o	o	o	o	o	x
10	x	x	x	x	o	x	o	o	o	o	o	x	x	x	x	x	x	x	x	o	x	o	x	x	o	x	x	x	o	x	x	o	x	x	x
11	x	x	x	x	o	x	o	x	x	x	o	x	o	x	o	x	o	o	x	o	x	x	o	o	o	x	x	x	o	x	x	x	x	x	x
12	x	x	x	o	o	o	o	x	o	o	o	x	o	x	x	x	x	o	x	o	x	x	x	x	o	x	x	x	o	x	x	x	x	x	x
13	x	x	x	x	o	x	o	x	o	x	o	x	o	x	x	x	x	o	x	o	x	x	x	x	o	o	x	o	o	x	x	o	x	x	x
14	x	x	x	x	x	x	o	x	o	x	o	x	o	x	x	x	x	o	x	o	o	o	x	x	o	x	x	x	x	x	x	o	o	o	o
15	x	x	x	x	x	x	o	x	o	x	o	o	o	x	o	o	o	o	x	o	x	x	x	x	o	x	x	x	x	x	x	o	x	x	x
16	x	x	x	x	x	x	o	x	o	x	x	x	o	x	o	x	x	x	x	o	x	x	x	x	o	x	x	x	x	x	x	o	x	x	x
17	x	x	x	x	x	x	x	x	o	o	o	o	o	x	o	o	o	o	o	o	o	x	x	x	o	o	o	o	o	o	o	o	x	x	x
18	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

Items Sheet	

Name	x	y	Image Name
Tickets	7	1	tickets.png
Tickets	21	1	tickets.png
Tickets	31	1	tickets.png
Tickets	32	6	tickets.png
Tickets	25	3	tickets.png
Tickets	19	5	tickets.png
Tickets	12	5	tickets.png
Tickets	17	11	tickets.png
Tickets	27	17	tickets.png
Tickets	19	3	tickets.png
Tickets	17	9	tickets.png
WetFloorSign	3	1	wetfloorsign.png
Popcorn	4	2	popcorn.png
Computer	34	14	computer.png

Using Reactors to Create Games

Remember the robot landing pad program from Lab 2? The position of the robot was captured with a coordinate: on each tick of time, the `on-tick` function produced a new y-coordinate value. The landing program also had a function to draw the robot based on the current coordinate.

All animations in Pyret have a similar structure: there is a datatype that captures the current information for the animation (positions, colors, attributes of characters, etc), and a separate function that takes a value of that datatype and draws it. Pyret calls these functions alternately: draw the current information, update it, draw it, update it, draw it, and so on.

For games based on user interaction, we want the information to update when the user presses keys (or the mouse). Rather than using the `on-tick` from the landing program, we use a different reactor function called `on-key`, which takes the game information and name of key pressed (as a string) and produces new game information. In this case, Pyret alternates between calling `on-key`, `to-draw`, `on-key`, `to-draw`, etc.

Thus, to make a game, you need a datatype that captures the information that changes as the game runs, you need a function that computes a revised datatype based on keys pressed, and you need a function that draws the current information. Pyret does the work of calling these functions to run your game.

You have seen the reactor properties `init`, `to-draw`, `on-key`, `stop-when`, and `close-on-stop` before. If you are implementing portals, you will also need to use the `on-mouse` function, which works similar to the `on-key` function; check out the `on-mouse` documentation (https://www.pyret.org/docs/latest/reactors.html#%28part._on-mouse%29) for more. (for this, Pyret adds `on-mouse` to the rotation of functions it calls.)

You can refresh yourself on reactor properties here (https://www.pyret.org/docs/latest/reactors.html). You may also want to look at your code or the solutions to Lab 2 (http://cs.brown.edu/courses/csci0111/spring2022/labs.html), which was on reactors.

Support Code

You will be given support code that provides four functions:

1. `load-texture :: (String -> Image)`

   This function will load the images you can use to build up your game. Given a name of a file from the project 2 image directory (https://cs.brown.edu/courses/csci0111/spring2022/assets/data/project-2/), `load-texture` will return that image. The textures (walls, empty squares) are 30×30 (pixels), and everything else (players, gadgets, portals, etc.) are 24×24 (pixels).

   Examples:

   - `load-texture("popcorn.png")` returns this `Image`:

   - `load-texture("computer.png")` returns

2. `load-items :: (String -> Table)`

   This function, given the ID of a spreadsheet, loads the `"items"` sheet into a table with columns `"name"`, `"x"`, `"y"`, and `"url"`.

3. `load-maze :: (String -> List<List<String>>)`

   and

   `load-maze-n :: (String, Number -> List<List<String>>)`

   This function, given the ID of a spreadsheet, loads the maze into a list of lists of the letters in the `maze-layout` sheet. If you change the number of columns, you must use `load-maze-n` which is the same as `load-maze` except it has a second

parameter which is the number of columns to load.
The resulting list looks like this:

Image Not Showing
Possible Reasons

- The image file may be corrupted
- The server hosting the image is unavailable
- The image path is incorrect
- The image format is not supported

Learn More → (https://hackmd.io/@docs/insert-image-in-team-note?utm_source=note&utm_medium=error-msg)

4. `get-maze-index :: (Number -> Number)` **(portals only)**
   The reactor `on-mouse` function takes in x and y coordinates that do not directly correspond to which row and column in the maze the user clicked on. Because of this, `get-maze-index` is a function that takes in the coordinate that is input to `on-mouse` and converts it to a game grid coordinate. For example, if the `x`-coordinate clicked is 334, `get-maze-index(334)` returns 11 (for the 12th column).

Design Check

For design check, you will hand in your `mildaflix.arr` and `pen-and-paper.pdf` file.

There are two types of questions for the Design Check: "Pen & Paper" and "Coding". For the Pen & Paper questions, come to your design check prepared to talk about your answers in the `pen-and-paper.pdf` file. For the Coding questions, write the code in `mildaflix.arr`. Submit both files to Gradescope at least 12 hours before your design check starts.

We recommend doing the questions in order as much as possible, as each question will help you think about the next.

Requirements

0. Find a partner for your project. **You may not work with your Project 1 partner**. (https://edstem.org/us/courses/12806/discussion/736312 (https://edstem.org/us/courses/12806/discussion/736312)).
    - fill out (once per pair) this Google form (https://forms.gle/5wSqP7F8b4pqg13s6) to let us know who your partner is.
    - **You must complete both by Wednesday, March 23rd at 11:59PM EST. If you do not submit the form by then, you will not be assigned a design check!**

1. **Pen & Paper**: Decide which item you will be implementing (gadgets or portals).

2. **Setup & Coding**:

    1. a) Make a copy of a configuration spreadsheet. In Google Sheets, you can do this by clicking `File > Make a Copy...`
       b) Share the spreadsheet with your partner, and
       c) give "Anyone with the link" the ability to view the spreadsheet (you have to go into the Advanced sharing menu to do this).
    2. Make a new Pyret file called `mildaflix.arr`.
       a) Copy in the following code, b) insert the ID of your spreadsheet, and c)

make sure it runs. Then look at `maze-data` and `item-data` to get an idea of what they look like.

```
# load the project support code
include shared-gdrive("dcic-2021", "1wyQZj_L0qqV9Ekgr9au6RX2iqt2Ga8Ep")

include shared-gdrive("project2-spring2022-support", "1ORus_ohkL48PWp-ffwKD

include image
include tables
include reactors
import lists as L

ssid = "INSERT YOUR SPREADSHEET ID HERE"
maze-data = load-maze(ssid)
item-data = load-items(ssid)
#| un-comment when ready!
maze-game =
  reactor:
    init              : init-state,
    to-draw           : draw-game,
    # on-mouse        : mouse-click, # portals only
    on-key            : key-pressed,
    # stop-when       : game-complete, # [up to you]
    # close-when-stop : true, # [up to you]
    title             : "MildaFlix"
  end
|#
```

3. **Pen & Paper**: In Lab 2 (https://hackmd.io/a9PQjXS2SwKukgnKF_8O3w), you made a reactor of a robot. Go back to that animation and look it over. Here is a table of the "Dynamic" (changing throughout the animation) and "Static" (does not change throughout the animation) components of the animation.

Static	**Dynamic**
Blue background	Robot height
Robot size	
Landing pad	

Look at the gameplay video at the top of the document. Make a similar list of the static and dynamic components of the maze game.

4. **Coding**: Figure out the datatypes you will need for this game. Keep in mind what you wrote for question 3: only "dynamic" information should be components of the game state.
   - Provide a datatype (called `GameState`) for the game state.
     - Note: You will need other datatypes for the components of the `GameState`.
   - For each new datatype you make, give some examples. In particular, create an example value of your `GameState` type.
     Put the data types and examples into `mildaflix.arr`.

5. **Pen & Paper**: The reactor `to-draw` function takes in the current `GameState` value and outputs an Image showing the corresponding frame of the game. Make a list of the tasks required to generate the maze image (3-5 one sentence bullet points). We suggest you spend a decent amount of time on this question: the more thorough you are, the more your TA will be able to help you at design check and the easier your jobs will be for week 2 of the project.

6. **Pen & Paper**:

(1) If you are working on the gadgets version of the game: The reactor `on-key` function takes in the current `GameState` value and outputs a new `GameState` value that reflects changes based on which key has been pressed. Pick one of the four keys (`W` , `A` , `S` , `D`) and write out the tasks (3-5 one sentence bullet points) needed to compute the new `GameState` based on the current `GameState` .

(2) If you are working on the portals version of the game: The reactor `on-mouse` function takes in the current `GameState` value, the `x` and `y` value of the position of the mouse event (in the reactor), and a `String` that represents the name of the mouse event. Write out the tasks needed to compute the new `GameState` based on the current `GameState` .

(1, 2) All: Again, work here should save you considerable time in week 2.

7. **Coding**:

   (1) If you are working on the gadgets version of the game: Write one **simple** example for the function named `key-pressed` you will use as `on-key` in the reactor.

   (2) If you are working on the portals version of the game: Write one **simple** example for the function named `use-portal` you will use as `on-mouse` in the reactor.

   (1,2) All: Be sure to provide (1) what are the inputs to this function, (2) what is the output of the function, (3) what the function is doing, and (4) an example, something along the lines of:

   ```
   key-pressed(..., "w") is ...
   ```

   or

```
  use-portal(..., 35, 70, "button-down") is ...
```

8. **Coding**: Write a function `get-item(l :: List<List<String>>, x :: Number, y :: Number) -> String` in Pyret that takes in a List of Lists, and get the xth element from the yth list. Try to add some tests too.

9. **Pen & Paper**: Write down a list of 3-5 edge cases that you can think of in the program and how you plan to tackle these edge cases. Some questions to think about: What are the edge cases of the keys being pressed/communicated to the program? What are the edge cases that could happen with the stamina logic?

Implementation Phase

Work on One Feature at a Time

The key to tackling a larger assignment like this is to build the features up gradually. For example, get the game working on just basic functionality before adding more advanced features. What might that mean here?

1. Implement a version with just the walls (no gadgets or portals), such that the player can move around the maze while respecting walls.
2. Read in the items (popcorn, portals, etc) from the Google Sheet, store them in your data structure(s), and make them show up when you draw the GameState.
3. Get the items to disappear after a player visits their cells.
4. Make the game handle the items (modifying stamina or teleporting).

These phases correspond to the grading levels for the project (see the Grading section below). You can pass the project even if you don't get beyond phase 1.

We strongly recommend saving a copy of your file after you get each stage working, just in case you need to get back to your last stable version (practicing programmers do this all the time).

Grading

Testing

Since there are many functions that will be written for this project, we will only be requiring minimal testing for all functions that are not critical functions. In an ideal world, you would thoroughly test all functions; however, we would rather see you test one or two functions really well (to show you understand the concept) and others partially, rather than only doing partial testing or no testing at all for all functions, especially critical functions.

Because of this, **we will grade (1) minimal testing (>= 2 test cases) for non-critical functions that output things other than an Image, and (2) your thorough testing (try to hit as many tests of edge cases as possible!) for the function that moves the character** (the one you use as `on-key` , or `on-mouse`). Test this function thoroughly, covering as many cases as you can. Regarding your other functions, write as many tests as you think you need to be comfortable with it working - but include at least 2.

Your design, testing, and clarity grades will not be based on your functionality grade. Functionality will be weighted more heavily in your final grade for this project in particular. Here is what we're looking at when grading functionality:

Minimum Functionality:

- A character that moves in response to key presses.
- Player cannot pass through walls.
- Game stops/ends when the character reaches a certain destination.

Mid-tier functionality:

- All above.
- Gadgets or portals are rendered on the maze.
- If doing gadgets: your character must have stamina that depletes upon moving. The gadgets do not need to be functional for mid-tier functionality. Upon reaching 0 stamina, the game should end.
- If doing portals: upon picking up a portal, your character can teleport. There does not need to be a limit of the portal range or the number of times the player can teleport for mid-tier functionality.

For full functionality:

- All above.
- If doing gadgets: tickets bring the character to full stamina; wet floor signs bring the character to a low stamina; popcorn heals for some amount; items disappear when walked over.
- If doing portals: there is a limit to how far the character can teleport and how many times a portal can be used.

Reflection

This project was designed to give you practice with organizing and manipulating structured data. Answer the following questions about these topics in a block comment at the bottom of `mildaflix.arr`.

1. Our support code represented the maze layout (walls vs floors) as a list-of-lists of strings, rather than a table. What were the advantages and disadvantages of this choice?
2. In this project, you worked with the maze in three formats: the Google Sheet with the configuration of walls, the list-of-lists version, and the image itself. For each representation, briefly describe what it is good and bad for.
3. Describe one key insight that each partner gained about programming or data

organization from working on this project.

4. Describe one or two misconceptions or mistakes that you had to work through while doing the project.
5. State one or two followup questions that you have about programming or data organization after working on this project.

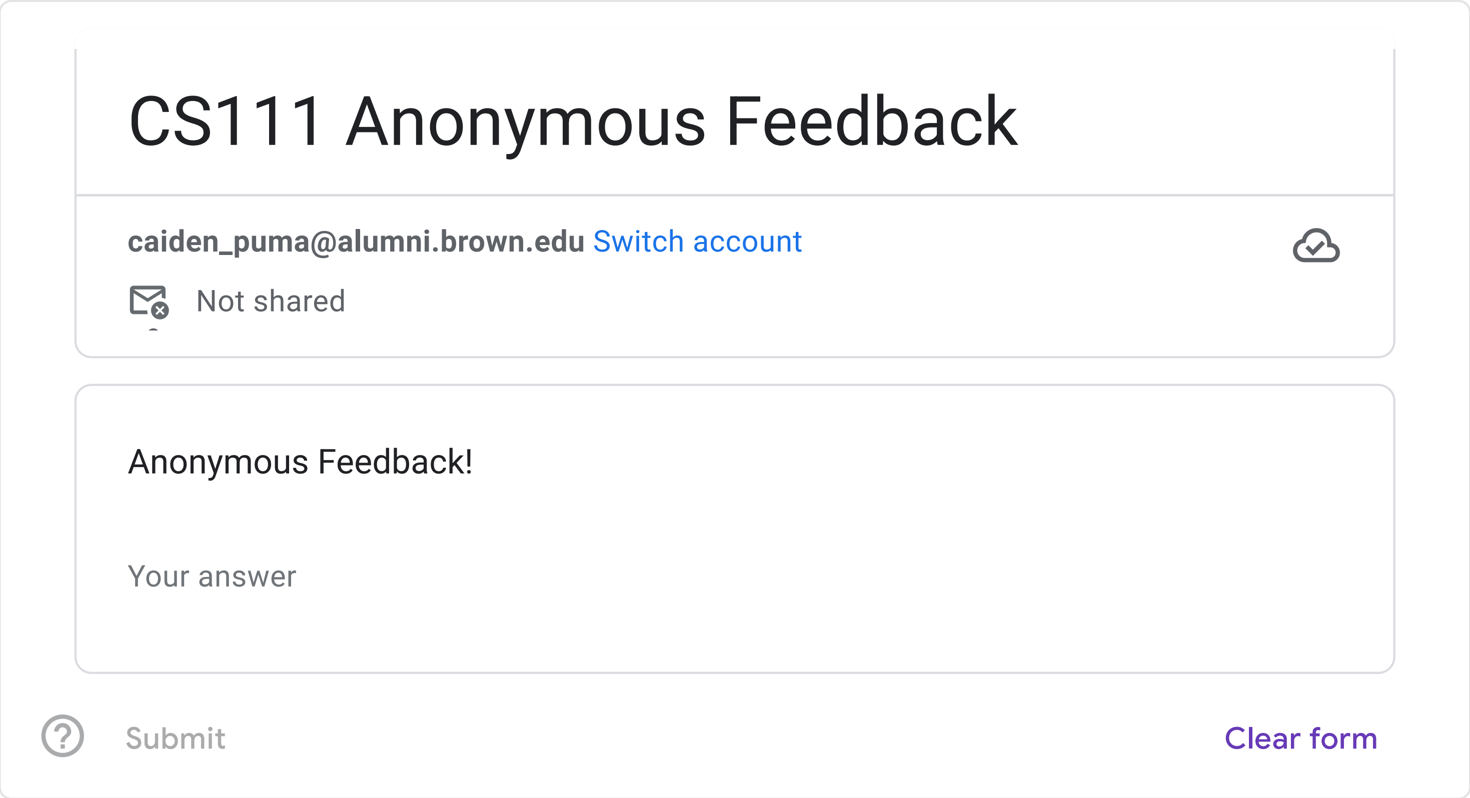
> Block comments are written with `#|` and `|#`:
>
> ```
> #|
>   this is a block comment.
>   with multiple lines!
>   yay!!
> |#
> ```

Handin

Hand in `mildaflix.arr` on Gradescope. Thank you to Gia F. for the amazing theming idea.

EdStem and Feedback

- EdStem (https://edstem.org/us/courses/18932/discussion/) can be your friend for this Project!



CS111 Anonymous Feedback

caiden_puma@alumni.brown.edu Switch account

Not shared

Anonymous Feedback!

Your answer