CSE 115
Introduction to Computer Science I
FINAL EXAM

Tuesday, December 11, 2018
7:15 PM - 10:15 PM
SOUTH CAMPUS

(Factor in travel time!!)
Room assignments will be published on last day of classes.
CONFLICT?

E-mail documentation to:

alphonce@buffalo.edu

Subject: CSE115 FINAL EXAM CONFLICT

no later than 5:00 PM TODAY
CONFLICT?

Alternate date/time

Friday December 14
11:45 AM - 2:45 PM
room TBA
Accessibility Resources

Schedule exam to overlap with alternate exam time

Friday December 14
11:45 AM - 2:45 PM
room TBA
EXAM RULES

Bring your UB Card

No arrivals after the first 30 minutes

No departures in first 30 minutes

No electronics
(phones, laptops, calculators, earbuds, etc)

Closed books. Closed notes. Closed neighbors
Road map

▶︎ CSE115 Game ◀
Game History

PreLab 3
• Movement behavior
• Maintain the game state as the player moves around the map

Lab Exam 3
• Use phaser.io to display the game in a browser
• Contained player movement, but was cut to fit the lab in 55 minutes

Today
• Showing the full game
Game Overview

Inspired by Chip's Challenge

Player must collect all crystals then find the end of the level

Collect keys to unlock doors

Avoid lava until the boots are collected
Map Tiles

Floor
• Player can move freely over floors

Wall
• Player cannot move through walls

Crystal
• All crystals must be collected to activate the portal

Door
• All doors are locked and require keys to unlock

Key
• Collect keys to unlock doors. Each key can only be used once
Map Tiles

Lava
• Will kill the player if not wearing boots

Boots
• Collect boots to walk freely over lava

Exit
• The exit portal for the level. Collect all crystals and enter the portal to complete the level

Player
• The player sprite
Building the Game

What will need to be built/coded to realize this game?
Building the Game

Some Sub-Problems

Represent Levels

Process player movement

Maintain the game state in memory (ex. number of crystals collected, how many keys does the player have, were the boots collected, where is the player)

Display the game UI

Host art assets
Representing Levels

Each tile type is represented by an integer:

- FLOOR: 0
- WALL: 1
- CRYSTAL: 2
- DOOR: 3
- KEY: 4
- LAVA: 5
- BOOTS: 6
- EXIT: 7

A level is represented by a 2d array of integers + starting location:

```
[1, 1, 1, 1, 1, 1, 1, 1],
[1, 6, 1, 1, 1, 1, 1, 1],
[1, 3, 1, 2, 0, 5, 7, 1],
[1, 0, 2, 4, 2, 1, 1, 1],
[1, 3, 1, 2, 0, 0, 0, 1],
[1, 2, 1, 0, 5, 5, 6, 1],
[1, 1, 1, 1, 1, 1, 1, 1]
```

start = [3, 6]
Representing Levels

The map is and grid of \((x, y)\) tiles with the origin at the upper left corner

To access the tile type of \((x, y)\) we use \(\text{map}[y][x]\)

\[
\text{map} = [

[1, 1, 1, 1, 1, 1, 1, 1],
[1, 6, 1, 1, 1, 1, 1, 1],
[1, 3, 1, 2, 0, 5, 7, 1],
[1, 0, 2, 4, 2, 1, 1, 1],
[1, 3, 1, 2, 0, 0, 0, 1],
[1, 2, 1, 0, 5, 5, 0, 1],
[1, 2, 1, 0, 5, 5, 6, 1],
[1, 1, 1, 1, 1, 1, 1, 1]
]
\]

\(\text{start} = [3, 6]\)
Player Movement and Game State

Coded in PreLab 3 (Full functionality in optional part 4)

As a player moves, track keys, crystals, and boots

If player moves to exit with all crystals, level is complete

If player moves to lava without boots, the level restarts

If player moves to a door with a key, the door opens and a key is removed from inventory
Using `phaser.io`

`phaser.io` is a powerful JavaScript game engine

We will only use it to display our game and to read keyboard inputs

Will use 3 primary functions
- preload
- create
- update
Using phaser.io

preload
• Called by phaser only once
• Used to load all the art assets into memory before starting the game

create
• Called by phaser only once and after preload is called
• Initializes the game itself

update
• Called by phaser every frame!
• Called 60 times per second if everything is running well
• Performance is critical. This function must run in less than 16ms
Using phaser.io

preload
create
update

• These functions will all be added to a game object

• Whenever one of these functions is called, we may want to refer to this game object

• To this end, we use the keyword this

• this refers to the object from which a function was called
function someFunction(){
    this.x = 7;
}

var obj = {
    fun: someFunction
}

obj.fun();

obj.x; // resolves to 7
Side Quest: this

this takes a while to understand

More exposure and explanation in CSE116

In our 3 phaser functions we will often use this to refer to the game object and update its values stored at certain keys

We also store a reference to the game object in a variable named game

Change made using this in one of the three phaser functions, or made using the game variable, will be reflected in both this and game
Using phaser.io

preload
- Preload images
  - this.load.image('0', 'floor.png');

create
- Initialize level 1
- Add key listeners
  - this.W = this.input.keyboard.addKey('W');

update
- React to user inputs
  - if (Phaser.Input.Keyboard.JustDown(this.W)) {} 
- Update the game state 
- Render the visuals to reflect the current game state
Using phaser.io

The update function must be efficient

In the code we are tracking the frame rate and displaying it on the console once per second so we can see if the frame rate drops

With only 16ms to run this function it's easy to drop frames
• Even adding console.log calls in this function can be enough to lag the game
  • Actually printing to the screen in any language is a very expensive operation

AAA titles accomplish a tremendous amount of rendering in these 16ms (or 33ms @ 30fps)
• This industry has pushed GPU technology and parallel processing to achieve high fidelity graphics within this time limit
Web Hosting

So far we've only focused on front end JavaScript for this game

We'll write a python web server to host our game

We can get away with preview in Codenvy for testing since Codenvy will run a server for us that will host our static files

If we want to run this outside of Codenvy and still have it available online we'll need to run our own server
Web Hosting

Wait, run this outside of Codenvy?

More to come on Wednesday
- Deploying a web app
Hosting Levels

Once our server is setup we will also host each level from python

This isn't strictly necessary since we could store all our levels in JavaScript, but it's good practice to store data on the server
- Enables user-generated maps
- Can write procedural generation algorithms in python and protect that code (i.e. generate new levels dynamically)

Levels are retrieved via AJAX calls when needed
Hosting Levels

Levels are retrieved via AJAX calls when needed

Since AJAX is asynchronous there will be frames that pass while the level is being retrieved

We handle this by having the update function return without updating the game while there is an AJAX call active

The problem arises because we have 2 of our functions running at the same time (JavaScript's update function and python's level function) and with a network delay
• How do we handle this in more complicated cases?
• Another CSE116 foreshadow..
Next Time

Deploying this game, or any web app, outside of Codenvy
Expansion Ideas

Music and sound effects

Create a level editor

Push blocks

Enemies

Track a player's current level even if they close their browser

A step counter with a leaderboard to show who completed each level in the fewest steps