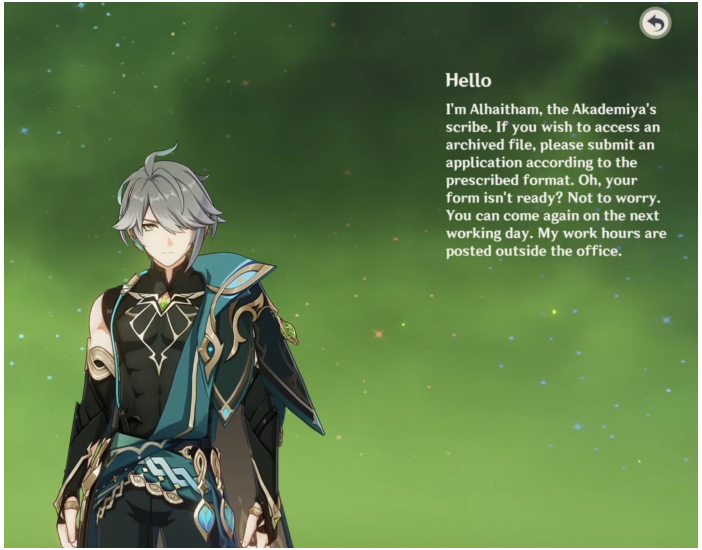


Lecture 37

Software Engineering IV

CS61B, Spring 2024 @ UC Berkeley

Circle Chen

About Myself - Liangyuan (Circle) Chen

- I am a junior studying CS and DS.
- I’m from Shenzhen, China.
- This will be my 6-th time teaching this course (including twice in the summer).
- I create and play games, and enjoy listening to game soundtracks.
- Itch profile: https://circlecly.itch.io/

Game Development

Lecture 37, CS61B, Spring 2024

Indie Game Development

- Almost every introductory CS course involves building games.
  - CS 61A (Hog, Ants)
  - CS 61B (2048, BYOW)
- We do this because games are fun.
- What if you liked making games and want to make this a hobby?
- Today I'll go over what indie game development is like.
- Indie means "independent".
  - Contrast to working at game companies like Riot and Blizzard.
  - A lot more freedom, but can still be demanding.
- Slide color background:
  - Blue slides might be useful for BYOW.
  - White slides are important for future game development.

Designing A Good Game

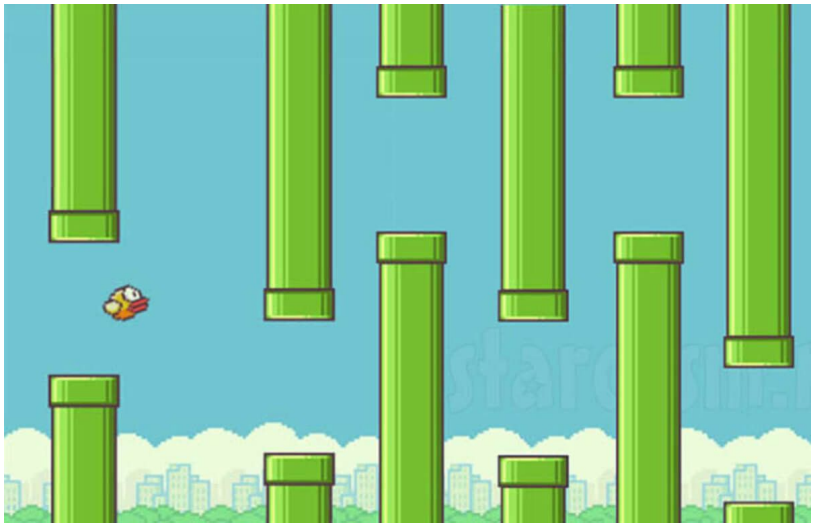
Lecture 37, CS61B, Spring 2024

How to design a good game?

- In Project 3 Phase A & B, you created random worlds and enabled the player to move in the world using keyboards.
- Project 3 Phase C involves adding features to make it more like a game.
- The key question: how should you design a good game?
- A good game should engage the player …
  - or even be addictive.

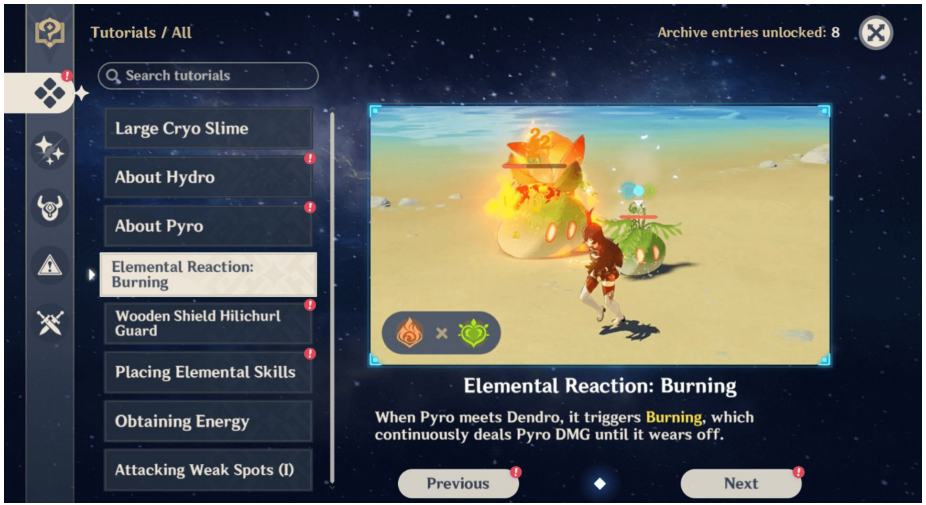
How to design a good game?

- Like Flappy Bird!
  - What the player can do: tap the screen (only 1 action)
  - What's difficult: very hard to time correctly.
  - Punchline: **easy to learn, but hard to master**.



How to design a good game?

- What if my game mechanism is inherently complicated?
  - The player should “learn” the mechanism gradually.
  - **Tutorial levels:** Explicit textual instructions given by the game.
  - May be embedded in an actual level, but the game still tells you what to do.



How to design a good game?

- The problem with tutorial levels:
  - Players don’t learn by reading text. They learn by interacting!
- “Show, don’t tell.”
  - **Antepiece:** an innocuous task that precedes a big challenge, that hints you at how you could tackle the big challenge.
  - No explicit verbal instructions, player should figure things out.
  - Example: Portal 2
    - Player can place two connecting portals to transport themselves.
    - Need to use portals to overcome obstacles.

How to design a good game?

- Can’t jump onto the platform!
- The orange portal is already created for you.
- If you place a blue portal on the wall, you’ll come out of the orange portal.
- This gives you enough velocity to reach the higher platform!
- Idea: A forward movement, out of a portal, across a gap, to a place you couldn't reach.
- A heuristic: “If you see a thrust-forward panel with an orange portal on it, try the trick you did in this level.”

How to design a good game?

- Now, the real puzzle.
- The player is likely to fall in the pit.
- They have to get out by a floor portal.
- This sets them up for the solution.
- Recall previous level:
  - “If you see a protruding panel with a portal, falling into another portal will thrust you out from there”
- Player will now try to jump into the blue portal intentionally.
- They’ll get thrusted to the other side!

How to design a good game?

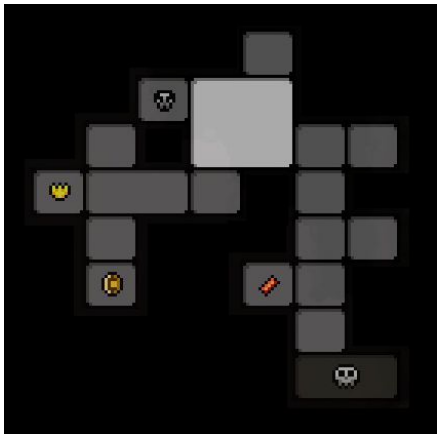
- Antepiece helps player learn… but why would the player want to learn?
- The player needs motivation to learn game mechanics.
- Hence we also need **feedback.**
- **Positive feedback:** when the player does well, they gain items, abilities, etc. so that the game becomes easier (and vice versa).
  - Reason: Players should be rewarded for doing well.
  - Ex. In most multiplayer combat games, the more opponents a player defeat, the more gold they get, and they can purchase more items.
  - What are some problems if positive feedback is too strong in a game?

How to design a good game?

- If positive feedback is too strong: players who are behind are very hard to catch up. A lot of focus on early-game decisions. Hence:
- **Negative feedback:** when the player does well, the game becomes harder, by giving them more challenging levels/worse items, etc. (and vice versa)
- Reason: Players should not fall too far behind. Things are more interesting if a lot of people are competing in the middle.
  - Ex. In some racing games, the last place player has higher chance get powerful boosts and weapons to knock out leading players.
  - What are some problems if negative feedback is too strong in a game?

How to design a good game?

- And speaking of motivation, there's one other problem: player knows everything after a successful run, so the game becomes boring.
- Hence, you need **variability of game experience**
  - You can **"randomize" each run** by randomizing items, rooms, etc.
  - Or you could just **keep adding new content.**



Interactive Question

- What elements would you add to BYOW to make it a good game?
  - Storyline - Immersion
  - Enemies / Obstacles
    - Something the player needs to overcome
  - Point System
    - Goal = maximize score?
    - Currency instead of points?

The Game Development Pipeline

Lecture 37, CS61B, Spring 2024

The pipeline for making a game

- Say you designed some good game mechanics. How do you actually turn it into the final product?
- You need to work with people on your team:
  - Designers
    - High level idea, level-design, plot, etc.
  - Artists / Musicians
    - Animations, sprites, music, sound effects (SFX)
  - Programmers
    - Code everything up
  - Testers
    - Find glitches in the game

The pipeline for making a game

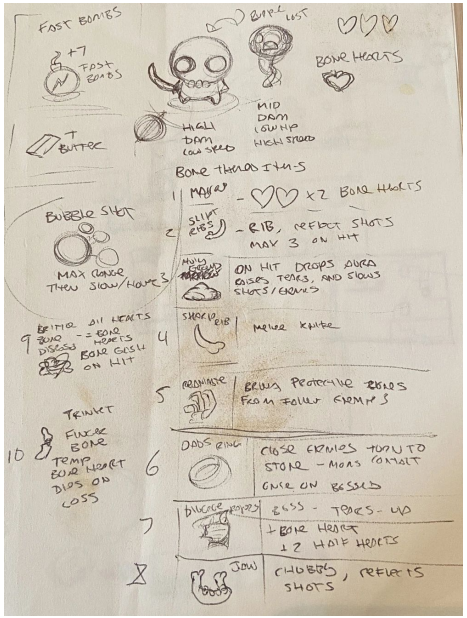
- The general process is:
  - You start with an idea.
  - Then you create paper-prototypes.
  - Then you create a Minimum Viable Product (MVP).
  - You iteratively improve your game, going through alpha -> beta -> release.
- We'll take Minecraft as an example.

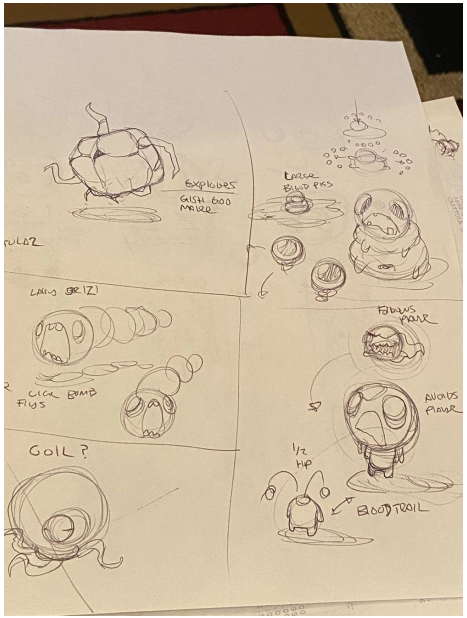
The pipeline for making a game - Idea

- The ideation phase is a couple sentences or elements relevant to the game.
- For Minecraft, this includes:
  - A block-based world.
  - Creativity (the player can do whatever they want in the world).
  - Randomly generated worlds.

The pipeline for making a game - Paper Prototype

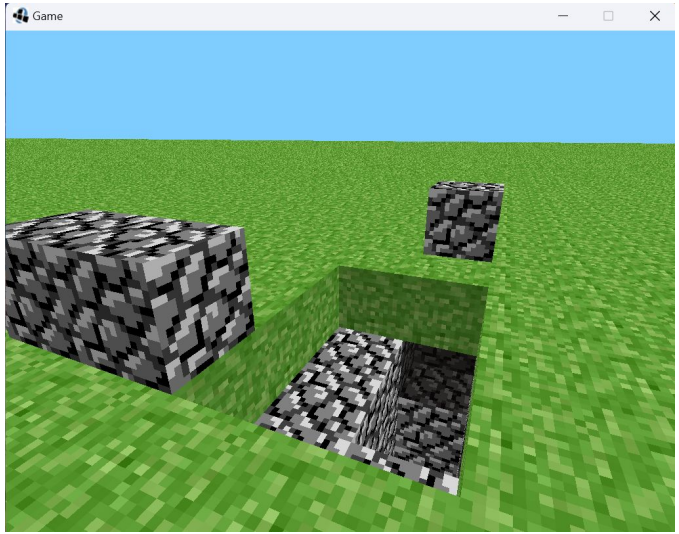
- A paper prototype describes the game on paper.
- The purpose: discuss, develop, and revise the details of the game internally without the need to write code.
- Couldn't find Minecraft paper prototype images, but here's a sample paper prototype for the game Binding of Isaac.





MVP (May 13 - 17, 2009)

- **Minimum Viable Product:** a bare-bones but playable version of the game.
- The purpose is to show “Hey, this idea works, I’ve implemented it in code!”
- For Minecraft: No monsters, no health, only grass blocks and cobblestone.
- WASD move around, Left click to place blocks, right click to destroy.
- Your BYOW would most likely top up at this stage.



Alpha (May 2009 - Dec 2010)

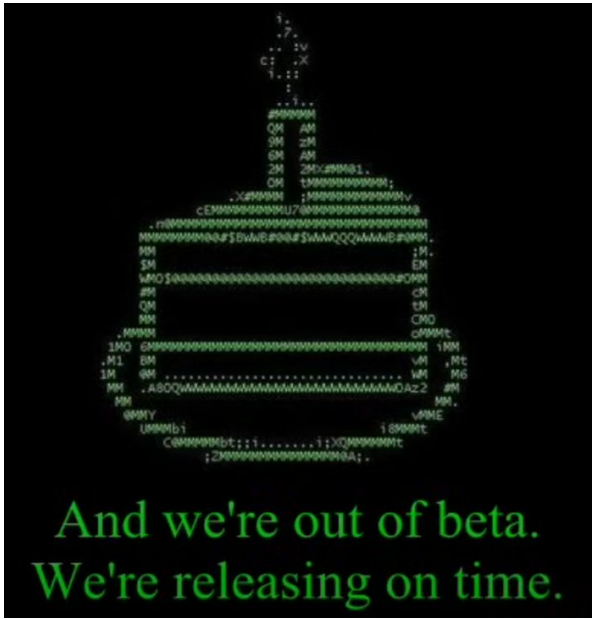
- The purpose of this stage is to show the game to a **larger audience than the developers**, including friends and other people you don't know.
- For Minecraft, this is when we have nether, multiplayer, fishing, etc.
- Still missing a lot of features from modern MC: notice no hunger bar below!

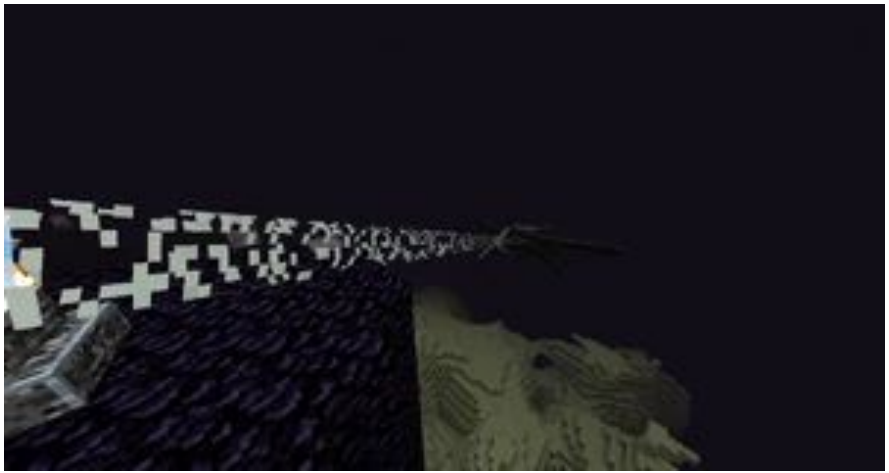
Beta (May 2010 - Nov 2011)

- The purpose of this game is a **preliminary release to the public**.
- For Minecraft, we now have both creative mode and survival mode.
- We have different weather, more biomes, and a hunger bar.

Full Release (Nov 18, 2011)

- At some point, you declare that the game is polished enough to be official.
- The cutoff between beta and release can be blurry.
- For MC, this is when you can defeat the Ender Dragon and “beat the game”!





After final release

- The game is still under development even after the official release.
- Developers polish the game, patch bugs, and gather user feedback.
- Minecraft receives version updates every now and then, when I started playing it was 1.4.7, now it is 1.20.2 (and it keeps going!).
- Contents like shulker boxes, netherites added in.

Minecraft Feedback > Knowledge Base > Release Changelogs

Minecraft: Java Edition - 1.20.2

We're now releasing 1.20.2 for Minecraft: Java Edition. This release comes with more diamond ore in the deep regions of the world and changes to mob attack reach as well as optimizations to the game's networking performance enabling smoother online play even on low-bandwidth connections.

This release also includes new features for map makers and pack creators like macro functions, a random command and pack overlays.

Programmer Perspective

Lecture 37, CS61B, Spring 2024

Coding up games from scratch

- We've talked about how games are made in general, but now we want to dive into how to code in games.
- The problem: design a version of BYOW with encounters, similar to encounter systems in Pokemon.

Let's revisit how the basic BYOW works:

- In a while loop, you would:
- Process user input (Checks for W, A, S, D)
- Update the game universe (Move avatars)
- Render the current state to the user (Calls `ter.renderFrame(tiles);`)
- You also wait for a bit until it is time to render the next frame.

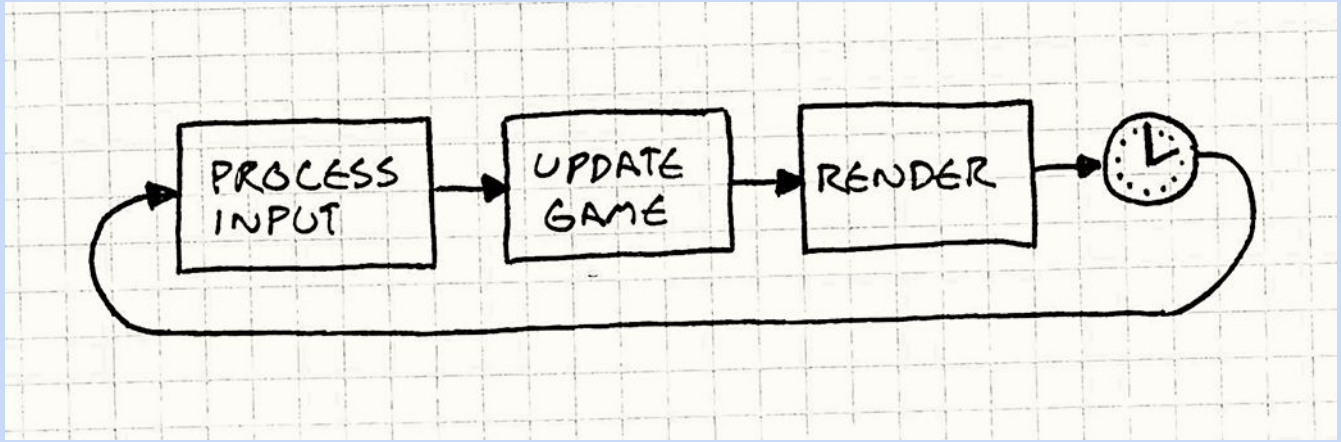


Image Credit: gameprogrammingpatterns.com

Coding up games from scratch - keeping the code clean

- In our BYoW with encounters: there exists different scenes.
  - Start at the main menu.
  - Press N -> “enter seed” scene.
  - Press S -> “world” scene.
  - “Encounter” scene.
- The Problem: Each scene will have different things to “update”.
  - If you are in the “world” scene, you render blocks and process WASD.
  - If you are in the “encounter” scene, you don’t want to render world tiles, but you render information pertinent to the encounter.

Coding up games from scratch - keeping the code clean

- Our goals:
  - Make game loop do different things in different scenes correctly.
  - Adding any new scene should be relatively easy.
  - Code structure should be as “neat” as possible.
    - Why? You will hate your past self if you don’t.
- How can we do this?

Coding up games from scratch - keeping the code clean

- Here's something that we don't want:

```
while (true) {
  // title screen
  char k = … //some method that gets the next key typed
  if (k == 'N') {

    // enter seed
    while (true) {
      k = … //gets the next key typed
      if (k == 'S') {

        // world movement
        while (true) {
          if (k == 'W') {

          } else …
        }
      }
    }
  }
}
```

Coding up games from scratch - keeping the code clean

- The issue: each new scene requires a new nested `while (true)`!
- Also, impossible to switch back and forth between three scenes!

```
while (true) {
  // title screen
  char k = … //some method that gets the next key typed
  if (k == ‘N’) {

    // enter seed
    while (true) {
      k = … //gets the next key typed
      if (k == ‘S’) {

        // world movement
        while (true) {
          if (k == ‘W’) {

          } else …
        }
      }
    }
  }
}
```

Coding up games from scratch - keeping the code clean

- Here's a better approach, but can still be improved:

```
// Represents current scene
private String current;

public void gameLoop() {
  while (true) {
    if (current.equals("title")) {

    } else if (current.equals("enterSeed")) {

    } else if (current.equals("world")) {

    } else if (current.equals("encounter") {

    } else if (current.equals("gameOver") {

    }
  }
}
```

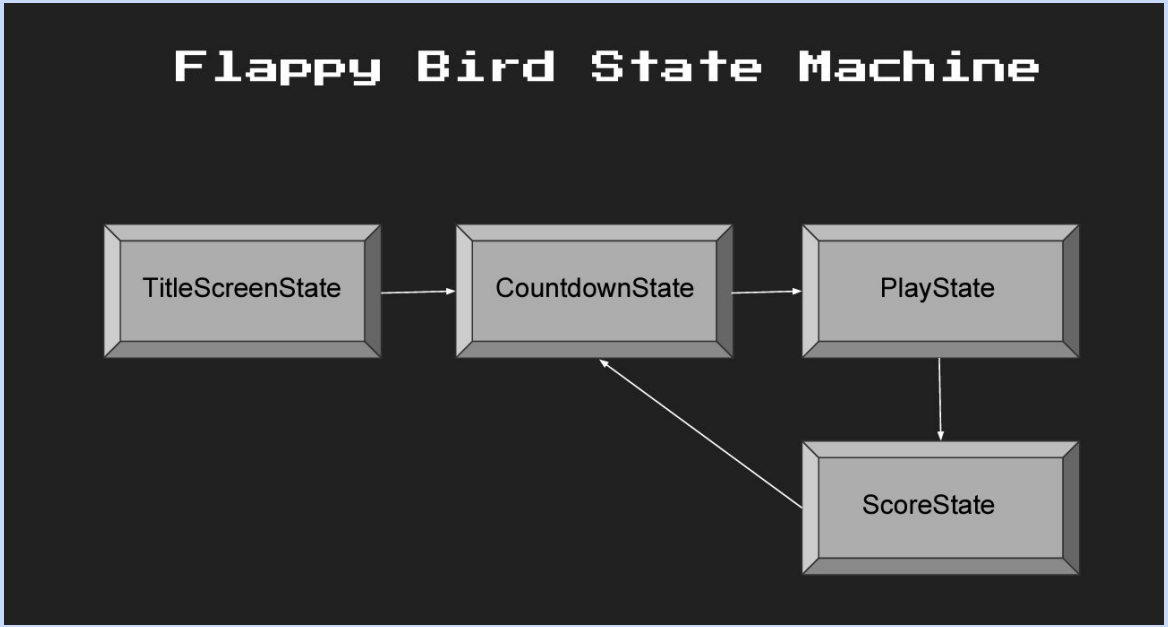
Coding up games from scratch - keeping the code clean

- Issues: game loop becomes longer with added scenes, hurts readability.

```
// Represents current scene
private String current;

public void gameLoop() {
  while (true) {
    if (current.equals(“title”)) {

    } else if (current.equals(“enterSeed”)) {

    } else if (current.equals(“world”)) {

    } else if (current.equals(“encounter”) {

    } else if (current.equals(“gameOver”) {

    }
  }
}
```

Coding up games from scratch - keeping the code clean

- We'll introduce the idea of a "finite state machine" to manage complex games with a lot of different scenes.

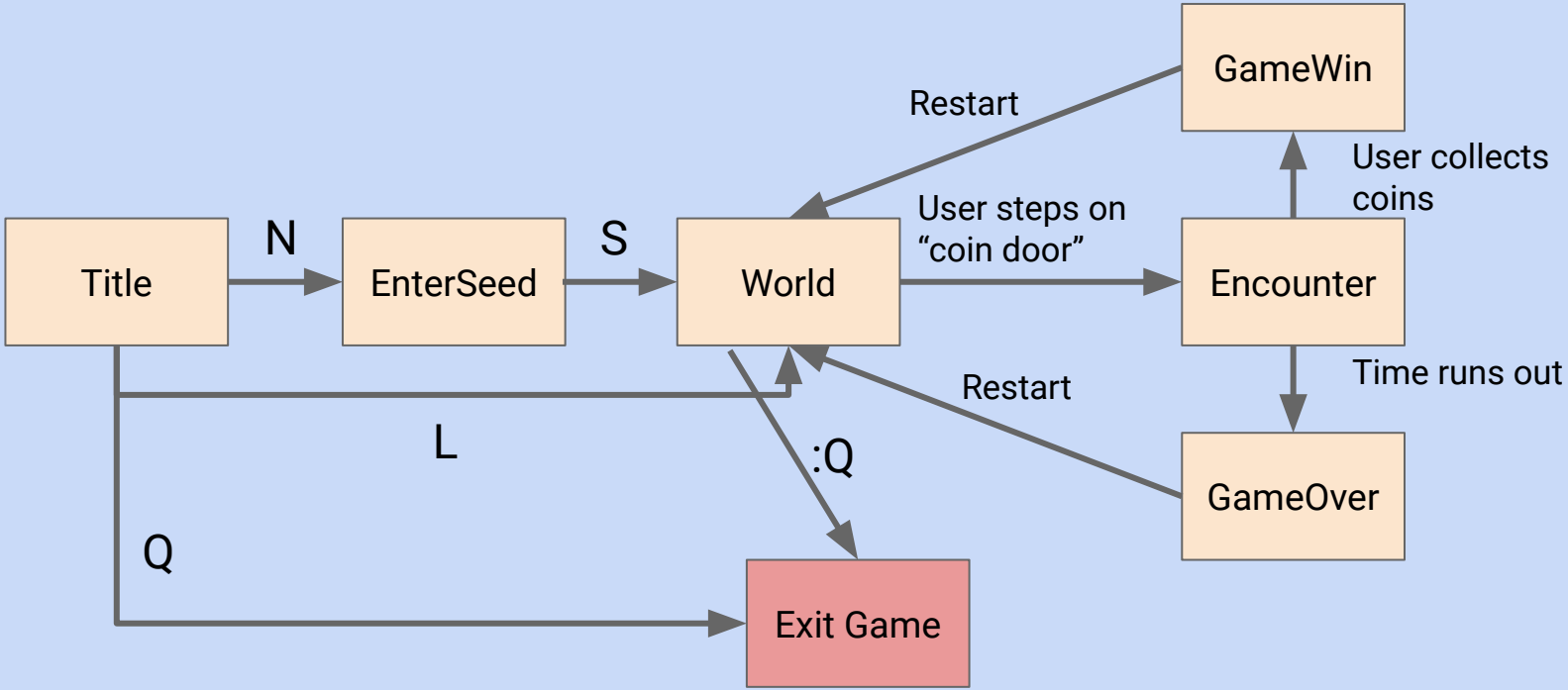


Coding up games from scratch - keeping the code clean

- A **finite state machine** is a lot like a directed graph:
- We have states that the machine can be in as “nodes”.
  - For instance, you will have Title, EnterSeed, World, Encounter, etc.
- We have transitions from one state to another as “edges”.
  - For instance, if you are in state “Title” and pushes L, you should go to state “World”. This corresponds to an edge from “Title” to “World”.
- The machine **can only be in one state at a time.**
  - Need a variable to keep track - this is extra from a directed graph!
- A sequence of inputs or events is sent to the machine.
- You check if any outgoing edges from your current state have their transition conditions fulfilled.

Coding up games from scratch - keeping the code clean

- Here's how BYoW could potentially be modeled by:
- This assumes you decide to do "win/lose", restart game without closing.



Coding up games from scratch - keeping the code clean

- Here's how you could implement an FSM in code:

```
// Represents current state
private StateMachine stateMachine;

public void gameLoop() {
  stateMachine = new StateMachine();
  while (true) {
    Set<Character> pressed = readInput();
    stateMachine.update(pressed);
    stateMachine.render();
  }
}
```

```
public class StateMachine {
  private HashMap<String, GameState> gameStates;
  public StateMachine() { /* Initialize "GameState"s */ }
  private String curr = "title";
  public void update(Set<Character> pressed) {
    gameStates.get(curr).update(pressed);
  }
  public void render() { gameStates.get(curr).render(); }
  public void changeState(String to) {
    // Throw errors if to is not valid
    gameStates.get(curr).exit();
    gameStates.get(to).enter();
  }
}

interface GameState {
  StateMachine mach; // the FSM I belong to
  void update(Set<Character> pressed);
  void render(), enter(), exit();
}
```

Coding up games from scratch - keeping the code clean

- And we can add new scenes as states…

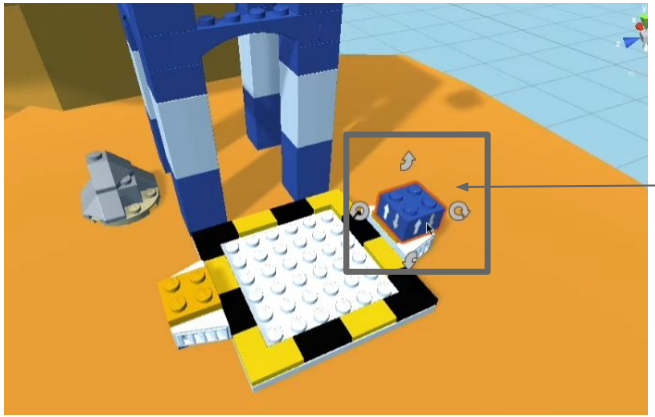
```
// Represents current state
private StateMachine stateMachine;

public void gameLoop() {
  stateMachine = new StateMachine();
  while (true) {
    Set<Character> pressed = readInput();
    stateMachine.update(pressed);
    stateMachine.render();
  }
}
```

```
// Each class in a separate file
interface GameState {
  StateMachine mach; // the FSM I belong to
  void update(Set<Character> pressed);
  void render(), enter(), exit();
}
public class WorldState implements GameState {
  public void update(Set<Character> pressed) {
    // Moves avatar based on “pressed”
    if (stepOnDoor())
      mach.changeState(“encounter”);
  }
  public void render() { /* draws the world */ }
}
public class EncounterState implements GameState {
  public void update(Set<Character> pressed) {
    // Moves avatar based on “pressed”
    if (wonEncounter())
      mach.changeState(“world”);
  }
  public void render() { /* draws objects in encounter */}
}
```

- By using FSMs, get a much cleaner main game loop!
- To add a new state, extend from `GameState`.

Game Engines Abstracts Away Details

- Coding games from scratch is hard. Requires code that exists in most games.
- Nowadays, people use **game engines** with all the boilerplate code built-in.
- One example game engine is called Unity.
- Unity is abstracted so well that you could develop games codelessly:
  - In Unity, every “thing” that exists in a game is a “game object”.
  - You can download game objects made by others from the Unity Store
  - Then drag and drop game objects in the Unity Editor!



When you attach this blue block to the platform, it will enable the platform to move up and down periodically.

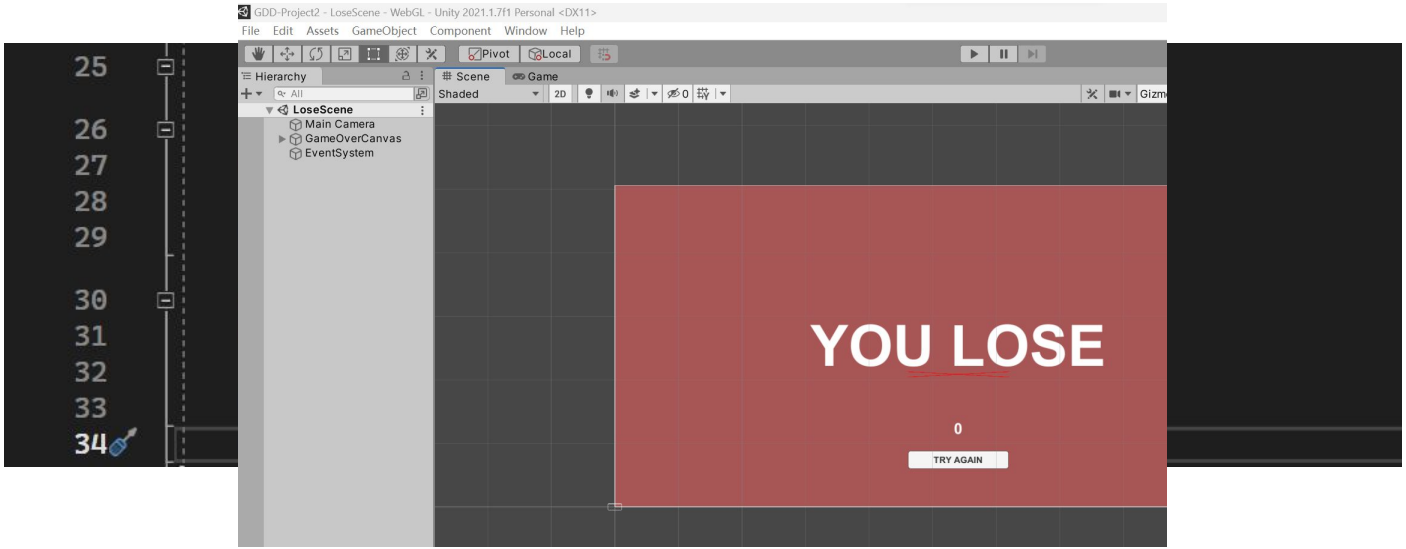
The code that does this is already written for you!

Of course, you can still write code

- Even if you decide to write code, the engine still abstracts a lot away for you.
- In BYOW, need to come up with class definitions for Java classes.
- But in Unity, you attach a “Script Component” to a game object that you want to control using code.
- The script should inherit from a class called **`MonoBehaviour`**, and it should override these methods:
  - **`void Start() // Called before the first frame update`**
  - **`void Update() // Called for every frame update`**
- The engine calls these methods behind the scenes for you.
- What about scene transition?
  - Unity has an FSM running behind the scenes!

Why Unity is simpler - Scene Transition

- In the Unity Editor, you can click on “File -> New Scene” to create a new scene for your game.
- You can define what game objects need to exist in that scene, and what they should do when the scene starts.
- When you want to transition between scenes, you just call **`SceneManager.LoadScene("another scene")`**!



But still, the ability to write good code matters…

- DRY (Don’t Repeat Yourself)
  - If you make different variants of an enemy, consider use inheritance.
  - e.g., green, blue, and purple slimes are all slimes, with different stats.
- Separate “data” from “logic”
  - Store enemy stats in a separate text file and load it.
  - Hardcoding is OK for small games, but bad for large games..
- Learn to work with different types of assets (resources used in games)
  - Images, audio, and 3D models.
- Need to know some algorithms
  - For instance, A* path finding for enemies.
- And finally, DRY (Don’t Repeat Yourself)

Game Modding

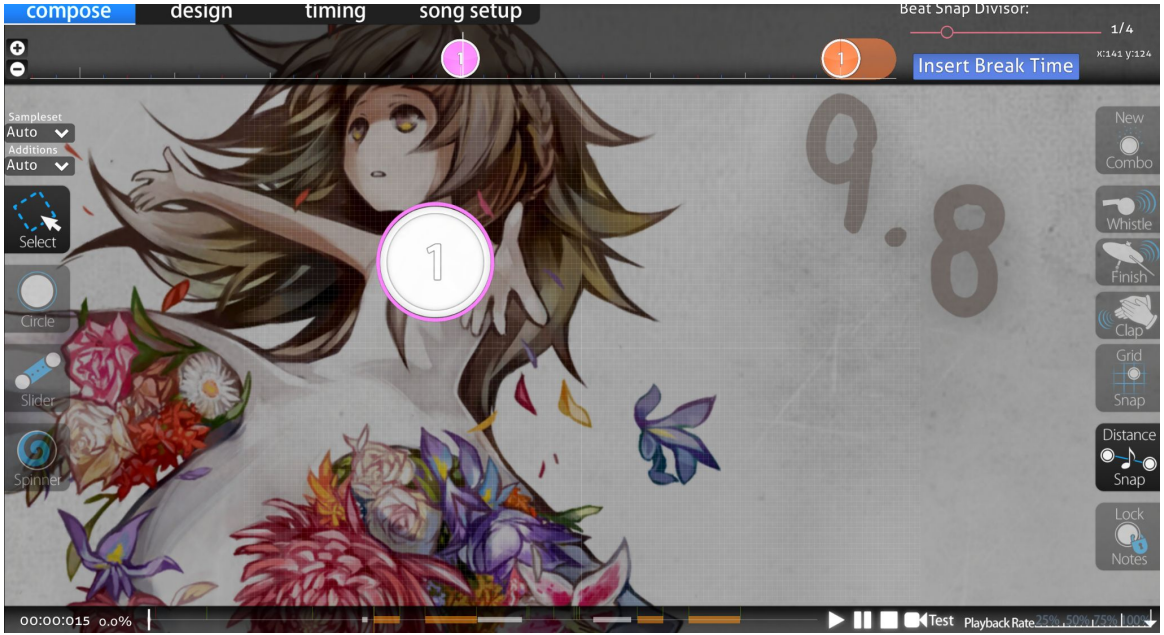
Lecture 37, CS61B, Spring 2024

Game Modding

- Let’s say you don’t want the trouble of creating an entirely new game, but just want to work off an existing game.
- This is called game modding.
  - “Mod” is short for “modify”.
- What is a “mod”?
  - An add-on to an existing game that provides new gameplay elements or modifies existing gameplay experience.

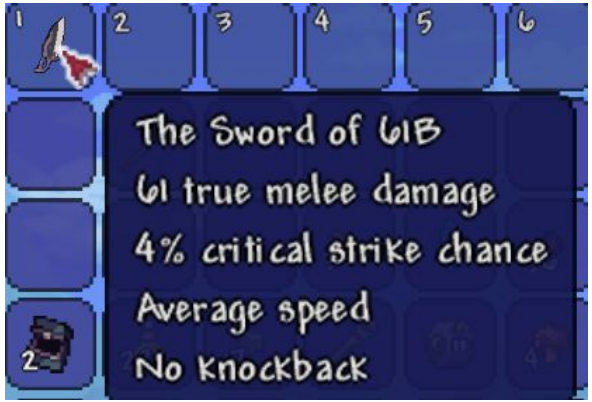
Game Modding - Adding new content

- One way to mod games is to add new levels.
  - For instance, osu!, a music game, allows creating custom levels via drag and drop. You can add songs as you wish.
  - Super Mario games also have a lot of player-made level editors.



Game Modding - Adding new content

- You can also choose to add new content using code (shown is Terraria modding). This would require reading tutorials and documentation.




```
public class Sword61B extends ModItem {

  @Override
  public void setDefaults() {
    item.damage = 61;
    item.damageType = DamageClass.MELEE;
    // Other logic
  }

  @Override
  public void OnHitNPC(Player player, NPC
target, NPC.HitInfo hit, int damageDone) {
    Main.NewText("Take that! The power of
Data Structures!");
  }
}
```

Game Modding - Is it legit?

- In the old days, a lot of games don't have official modding support.
  - Minecraft and Terraria didn't originally officially support mods.
  - But people in the community "reverse-engineer" games secretly and created mod support platforms for people to build mods.
  - Technically modding is grey-area, but the game companies realized that mods actually help with popularity of game and retain users.
  - Eventually modding became (semi-)officially supported.
  - **Bottom lines: never release the source code of the vanilla game!**
  - **Do not mod a game if the developer explicitly bans it.**
- For some games, official mod support is provided from the start.
  - Examples are Binding of Isaac; Don't Starve Together

Game Modding - How does modding affect the game itself?

- Game developers take away from good mods
- Example: Binding of Isaac Repentance update - draw a lot of content from Binding of Isaac Antibirth. It was the largest mod for Binding of Isaac at that time yet.
- Another example: Super Mario Maker
  - A lot of players creating own levels - why not create a game about that?

Repentance is a DLC expansion to *The Binding of Isaac: Afterbirth †* that was released on March 31, 2021.

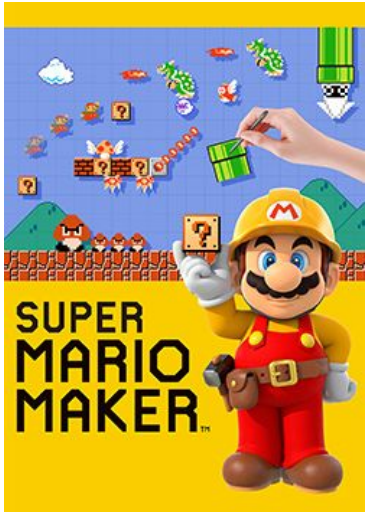
DLC!

For a list of the pre-release blog posts, see The Binding of Isaac: Repentance (Pre-Release).

Features

General features revealed include:

- The fan mod The Binding of Isaac: Antibirth integrated into the game as official content!



Summary

- Coming up with good game ideas.
  - Easy to learn, hard to master; Antepiece; Feedback; Variability
- The pipeline of making a game.
  - Idea, Paper Prototype, MVP, Alpha, Beta, Release
- Writing code for games
  - Using pure Java - why FSM is good
  - Using Unity - why game engines are good
- Modding an existing game.
- At each layer, you lose some freedom, but do less “dirty work”.
- But, you can choose whatever layer you enjoy the most when you develop an indie game - you’re not constrained by deadlines or company goals.
- Therefore, I hope you find this lecture and indie game development fun and would consider developing this as a hobby.

Q&A