Final Project Planning

1	6				5
		5	2		
5				3	
	4				1
		4	1		
3				5	4

Project Overview

Choose Your Game

You'll have a choice from five games to implement. Whichever game you choose, you will be responsible for implementing the game to the specifications that we provide.

Object-Oriented Implementation

Your project must use object-oriented design. If you implement your entire project as static, with no object instances, you will lose significant points.

Plan, then Program

We encourage you to spend time thinking out the design of your game before you begin programming it.

Step 0: Choose a Game

Choices: Sudoku

Rules:

- 6x6 grid of numbers, mostly empty to start
- Fill in blanks so that no number repeated in any row, column, or 2x3 box
- Implementation will need to read in starting boards

1	6				5
		5	2		
5				3	
	4				1
		4	1		
3				5	4

Choices: Brick Breaker

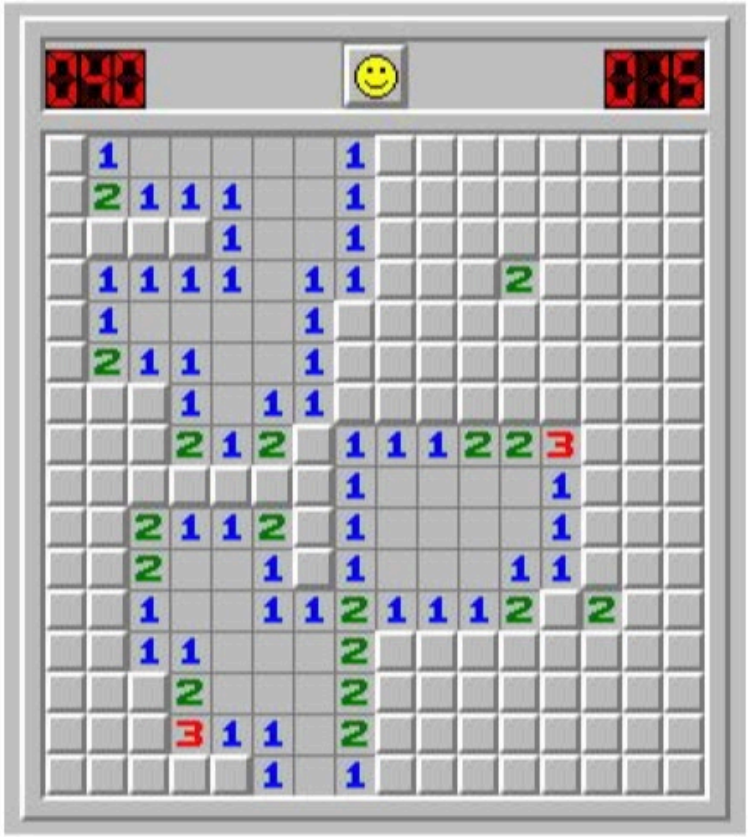
Rules:

- Clear a group of blocks by hitting them with a bouncing ball
- Ball is always moving, player controls paddle at bottom
- Bricks require multiple hits to break

Choices: Minesweeper

Rules:

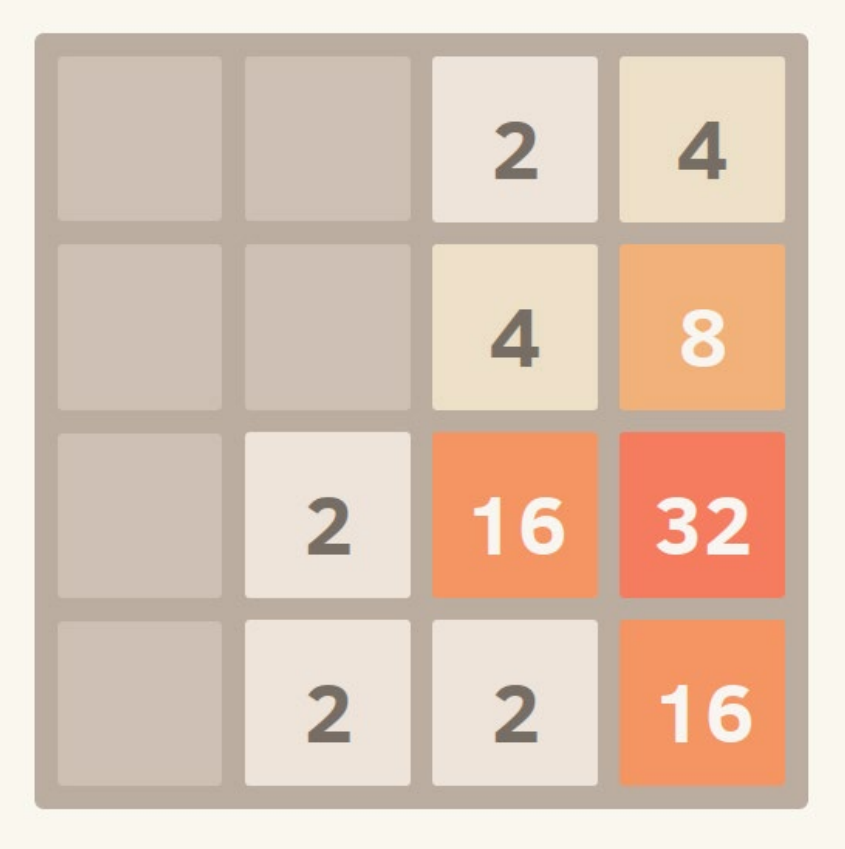
- Start with a board of hidden squares, each containing either a bomb or a number
- Clicking a bomb is game over
- A number square shows how many bombs are in adjacent squares



Choices: 2048

Rules:

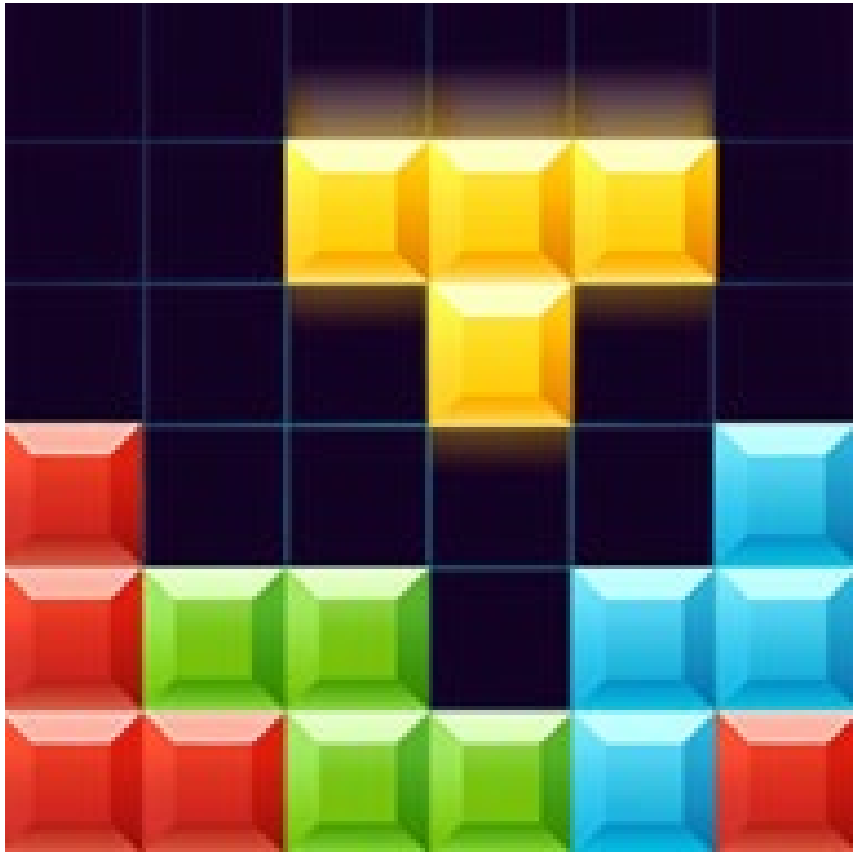
- Board is a series of squares; each is empty or containing a number
- Each turn, slide squares N, E, S, or W
- Squares of the same value combine (2 + 2 becomes 4)
- New 2 or 4 appears each turn; keep going until 2048



Choices: Tetris

Rules:

- Pieces consisting of four block combinations fall down a grid, one at at time.
- Filling a line with pieces clears the line, awards points
- Game over when piece settles at the top of the grid.



Step 1: Understand the Requirements

Play a Version of the Game Online

Links:

- Sudoku
- 2048
- Minesweeper
- Brick Breaker
- Tetris

Brick Breaker Requirements

Requirements

- The game must have a restart feature such that when the player has no more lives, the screen redirects to a page that requires user input to restart the game as if the program were run for the first time.
- Each time the ball misses the platform or if the game is being played for the first time, the game must begin with the ball stationary on the platform and require player input before the ball is allowed to begin movement again.
- The platform must only move in the x-direction.
- There must be at least three kinds of bricks with varying strengths and their strength should be differentiated with some visual cue.
    - Each of the three kinds could be a different color
    - Or you could display a number on each brick, representing its strength
- In addition, when a brick loses strength after a collision, there should be some visual cue to indicate that it was hit by the ball.
- The ball must bounce off the top, left, and right boundaries.
- The ball must move with some velocity in the x-direction and y-direction as well as an acceleration.
- The acceleration should change in different way when it collides with a brick compared to when it collides with the platform.
- If the player eliminates all of the bricks, the game should stop, a victory message should be displayed in text on the screen, and the player should have the option to start a new game.
- It is up to you how complex or simple to to make this movement, but at a minimum the acceleration should be affected in some unique way in the two cases described above.

Brick Breaker Requirements, Annotated

Requirements

- The game must have a restart feature such that when the player has no more lives, the screen redirects to a page that requires user input to restart the game as if the program were run for the first time.
- Each time the ball misses the platform or if the game is being played for the first time, the game must begin with the ball stationary on the platform and require player input before the ball is allowed to begin movement again.
- The platform must only move in the x-direction.
- There must be at least three kinds of bricks with varying strengths and their strength should be differentiated with some visual cue.
    - Each of the three kinds could be a different color
    - Or you could display a number on each brick, representing its strength
- In addition, when a brick loses strength after a collision, there should be some visual cue to indicate that it was hit by the ball.
- The ball must bounce off the top, left, and right boundaries.
- The ball must move with some velocity in the x-direction and y-direction as well as an acceleration.
- The acceleration should change in different way when it collides with a brick compared to when it collides with the platform.
- If the player eliminates all of the bricks, the game should stop, a victory message should be displayed in text on the screen, and the player should have the option to start a new game.
- It is up to you how complex or simple to to make this movement, but at a minimum the acceleration should be affected in some unique way in the two cases described above.

Step 2: Plan the Game's Entities

What are the important pieces of the game?

What do these pieces each do?

What properties do they have?

The Entities of Brick Breaker

Entities	The game/player	The ball	The bricks	The platform
Actions				
Properties				

Did I miss any entities?

Did I combine any that should be separate?

Do all of these entities make sense to include?

Step 3: Decide how the Entities Become Interfaces and Classes

Refresher on Interfaces and Classes

- Classes:
  - Define objects
  - Contain methods (actions) and fields (properties)
  - **Can be instantiated**
- Interfaces:
  - Define *Abstract Data Types* by specifying what implementing classes can do.
  - Contain abstract methods (function signatures) and constants (unchanging properties)
  - **Cannot directly be instantiated from**

How to Plan Classes & Interfaces

- Steps 1 & 2 teach you the rules of the game and the components that drive it.
- By now, you should have a table with a list of entities, their actions, and their properties.
  - The actions become the methods that the objects should perform
  - The properties become the fields that the objects contain
- Not every object will need to directly implement an interface
  - There is a natural way to incorporate at least one interface into each game
  - A good strategy is to write a simplified interface for every entity before coding.

Planning Interfaces for Minesweeper (Incomplete Table)

Entities	The game	???	Mine Cells	Number Cells	???
Actions			*appear on screen*, *reveal*, end game	*appear on screen*, *reveal*	
Properties			revealed?	revealed?, number neighboring bombs	

Planning Interfaces for Minesweeper (Key Takeaways)

- From even an incomplete look, we can observe that the two types of cells in the game have similarities in what they can do.
  - Clicking a hidden bomb ends the game. When revealed, the square should show a bomb.
  - Clicking a number cell lets the game proceed. When revealed, the square should show a number (or an empty square)

    We have the notion of a Cell, which is an abstract type that should be able to draw on screen and reveal itself when clicked.

The Cell Interface

```
public interface Cell {
    void draw();
    void reactToClick();
}
```

Classes Implementing Cell

```
public class BombCell implements Cell {
    public BombCell(Point position) {...}
    public void draw() {...}
    public void reactToClick() {...}
}

public class NumberCell implements Cell {
    public NumberCell(Point position)
{...}
    public void draw() {...}
    public void reactToClick() {...} //
show info
}
```

Classes Without Interfaces

- Not all classes need to implement an interface.
- The overarching `Game` class is a good example
  - You're only writing one type of Minesweeper
  - This might not be true if you made Minesweepers of varying difficulties or rules!
- Still, it is important to plan the classes like interfaces before you write.

The Game Class as Pseudo-Interface

```
Game.java
    boolean isGameOver
    int width, height
    ...
    void drawGame()
    boolean handleClick()
    void resetGame()
    ...
```

Step 4: Understanding the Game Model

Really, it's Step 4.1, 4.2, 4.3, …

When Should the Game State Update? (4.1)

- Should the game state update…
  - with every animation frame?
  - only when the player interacts?

When Should the Game State Update?

- Should the game state update…
  - **with every animation frame?**
  - only when the player interacts?
- *Continuously* updating games
  - Brick Breaker
  - Tetris

When Should the Game State Update?

- Should the game state update…
  - with every animation frame?
  - **only when the player interacts?**
- *Discrete* updating games
  - 2048
  - Minesweeper
  - Sudoku

High-Level Template for Continuous Animations using PennDraw

```
1 public class Game {
2     public static void main(String[] args) {
3         GameStateManager gsm = setup();
4         PennDraw.enableAnimation(30);
5 
6         while(true) {
7             gsm.draw();
8             gsm.update();
9 
10            if (gsm.checkEndCondition()) {
11              break;
12            }
13 
14            PennDraw.advanceAnimation();
15        }
16    }
17 }
18 
```

Adding Interaction Example (Continuous)

```java
public class Game {
    public static void main(String[] args) {
        GameStateManager gsm = setup();
        PennDraw.enableAnimation(30);

        while(true) {
            gsm.draw();
            gsm.update(PennDraw.mouseX(), PennDraw.mouseY());

            if (gsm.checkEndCondition()) {
              break;
            }

            PennDraw.advanceAnimation();
        }
    }
}

```

Saving Updates for after User Interaction (Making Discrete)

```java
public class Game {
    public static void main(String[] args) {
        GameStateManager gsm = setup();
        PennDraw.enableAnimation(30);

        while(true) {
            gsm.draw();

            if (PennDraw.mousePressed()) {
                gsm.update(PennDraw.mouseX(), PennDraw.mouseY());
            }

            if (gsm.checkEndCondition()) {
              break;
            }

            PennDraw.advanceAnimation();
        }
    }
}

```

What Data Structures Should I Use? (4.2)

- Reminder: data structures are types we use to contain multiple pieces of data.
- Examples from this semester:
  - Arrays
  - 2D Arrays
  - Lists

Using an Array

- Great for storing a fixed amount of data points
- `Color[] pieceColors = {new Color(255, 0, 0), new Color(0, 255, 0), new Color(0, 0, 255)}`
  - Perhaps for Tetris, you might want to choose the colors for each piece type.
  - Then, perhaps: `Piece nextPiece = new Piece(pieceColors[1])`

Using a 2D Array

- Excellent for representing a grid of things in rows and columns.
- For instance, in a game like 2048, we're working with a grid of 4x4 tiles.
  - `Tile[][] gameBoard = new Tile[4][4]`
  - This initializes a new board of tiles
  - Accessing the tile in row 0, column 2: `gameBoard[0][2]`

Using a List

- Great for storing a sequence of data that can grow and shrink
- Possible uses:
  - Store the Bricks to break in a randomly initialized List
  - Track all dropped Tetris pieces played in game so far
    - Could grow to be quite large!

What's the Ending Condition of the Game? (4.3)

- We specify each of the ending conditions, so you don't have to think too hard…
  - e.g. 2048: when the board is full, or the user generates a 2048 tile.
- What do you do after the game ends?
  - Some of our requirements involve resetting the game
  - Some should just show an ending screen and stop

When do I Start Writing Code? (4.4)

- Short answer: Now
- Longer answer: Now, but you'll often retrace your steps
  1. "What should I do when this hits that?"
  2. "It looks like the Board and the Game should really be different entities…"
  3. "There should actually be two separate methods for checking collisions and handling them."
  4. "These two types of obstacles should both inherit from the same interface"
  5. "I don't understand how I'll check the end state using this data structure."

A Worked Example

Requirements

- A dinosaur runs to the right, jumping over obstacles
  - Two types of obstacles: cacti and pterodactyls
- Press SPACE to jump
- Touching an obstacle leads to death
- Accumulate points continuously while alive

Planning Entities

Entities	The game/player	_____	_____	_____	_____
Actions					
Properties					

Deciding on Classes and Interfaces

Discrete or Continuous?

What Data Structures?

Ending Condition?

Start Writing Code

WEDNESDAY