

CS 91S Game Systems
Fall 2024



Jellybean Chase

CS91: MAKE project

Sumin Byun
Stephanie Kim

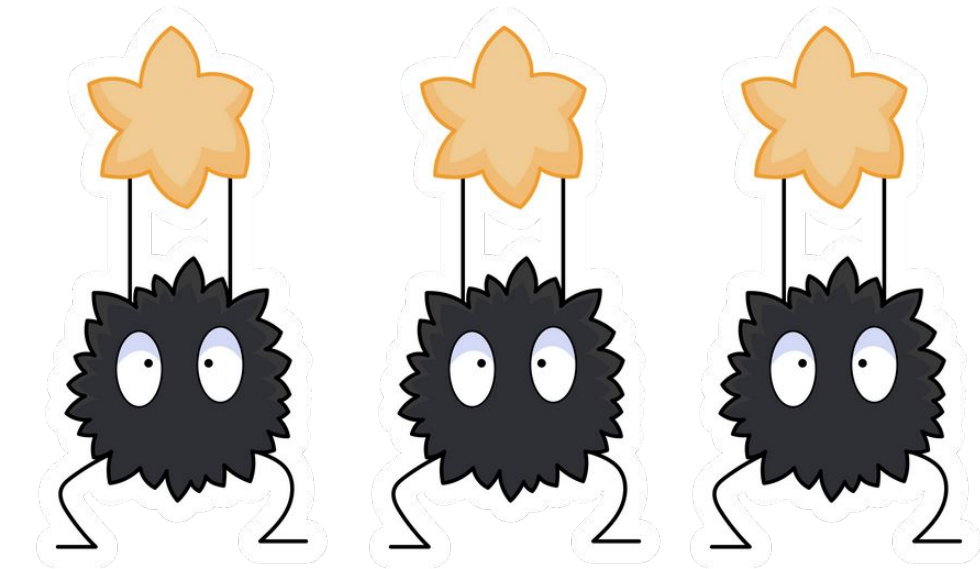
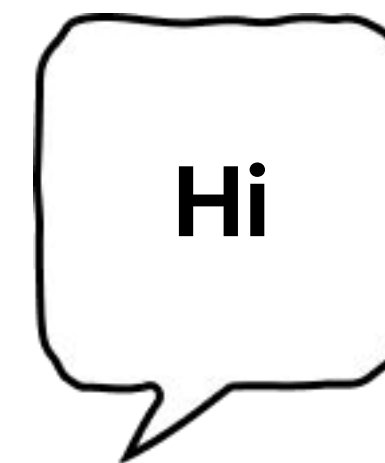
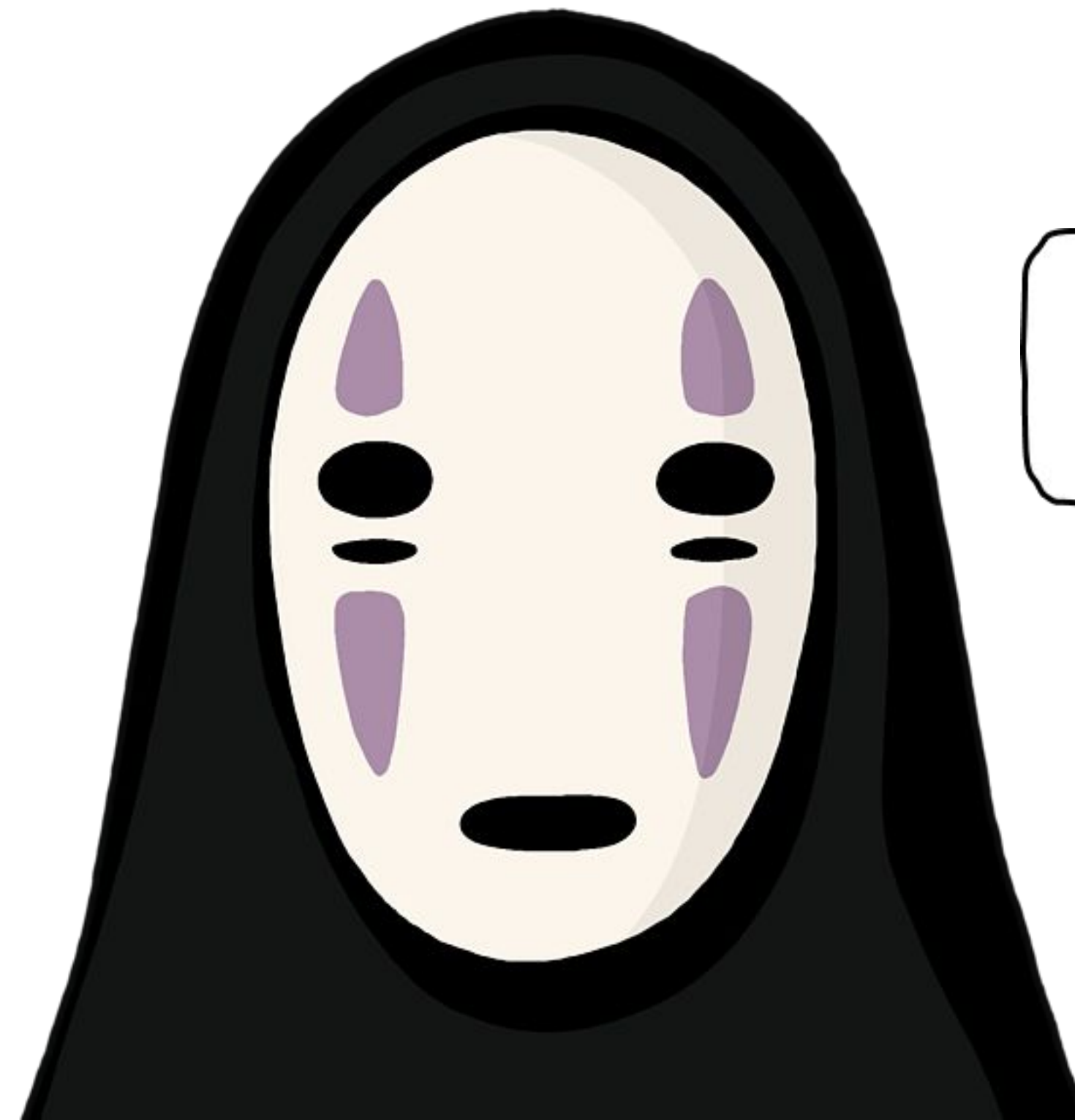
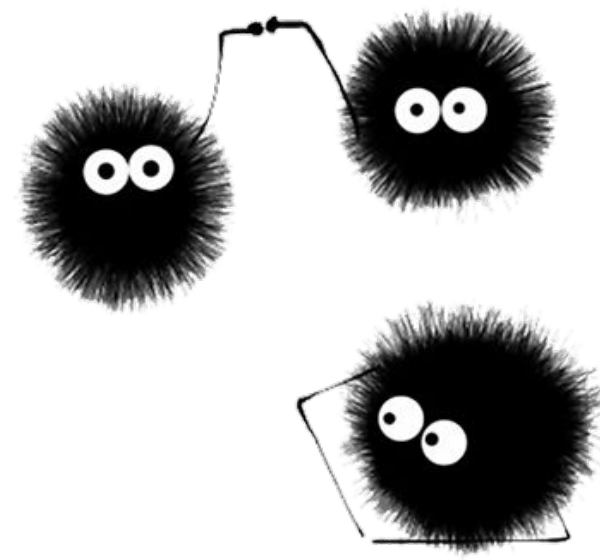


Table of Contents



01

Introduction

Introduce Our Game

02

Game Physics

Movement and Collision

03

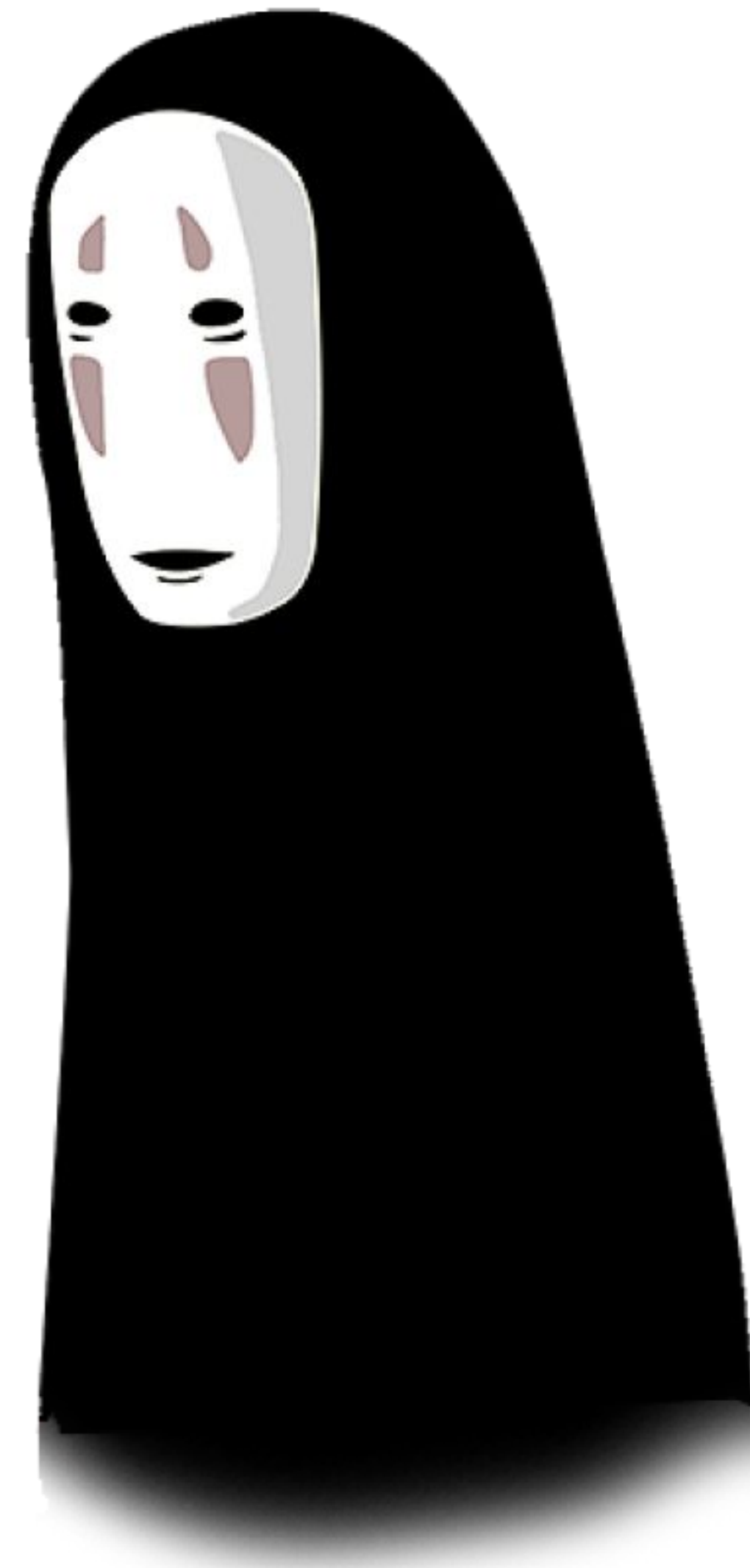
Design Patterns

Middleclass, Flyweight, Game loop,
State, Object pool

04

Overview

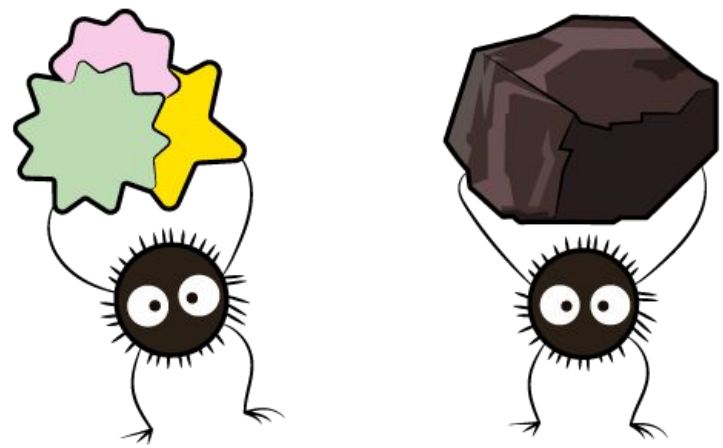
Game Link



Game Play



Move the bird to eat all 6 jelly beans
as fast as possible without hitting
any of the floating heads!



How to Play

Studio Ghibli Theme



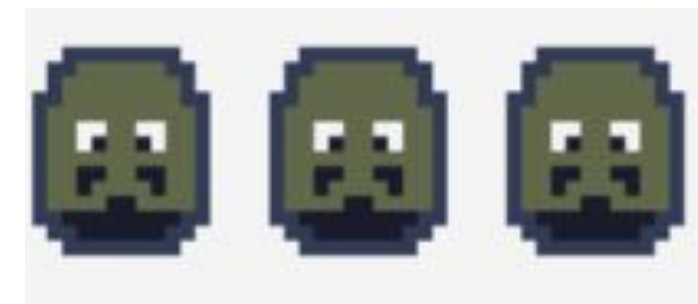
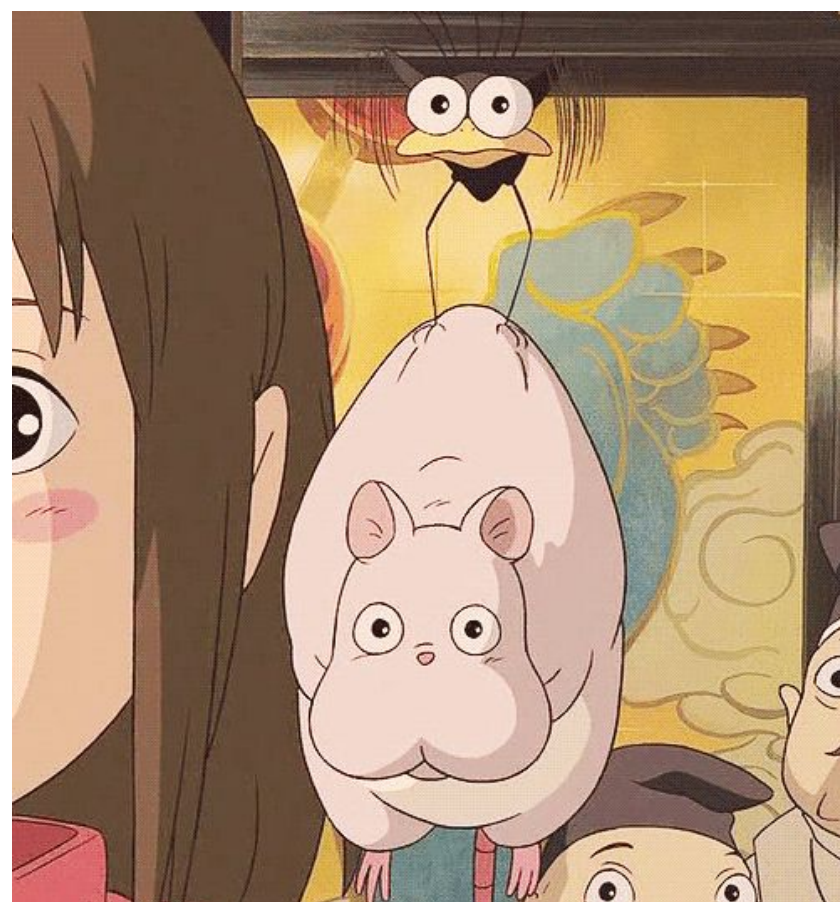
Player

Move using arrow keys



our character

original character



our character

original character



Eat Jelly Beans

Bounces around screen with randomized direction.



our character

Avoid Obstacles

3 horizontally-moving and 3 vertically-moving.
Player dies when hit!



original character

Game Physics



Movement

Always boundary checking

Player: vector operations based on user input of arrow keys

Targets: vx and vy randomly assigned every 80 frame counts (~1.33s) to be either -1 or 1

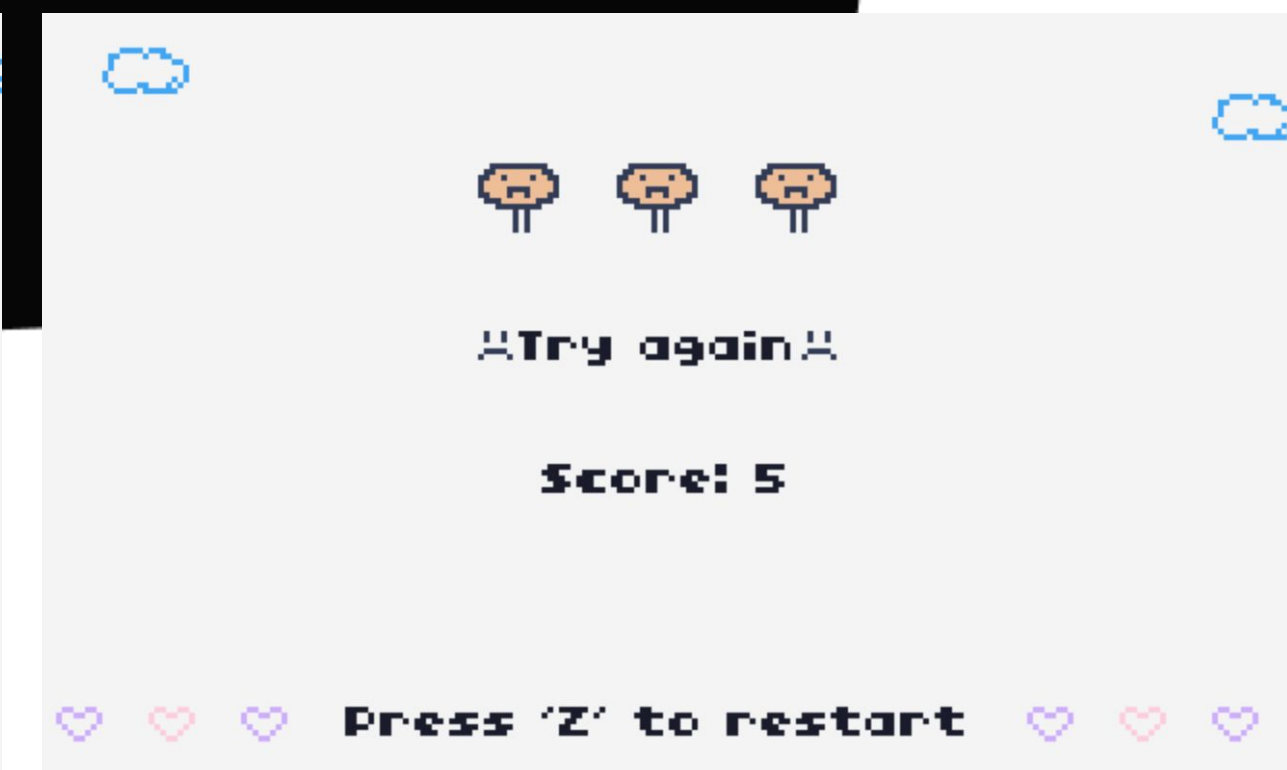
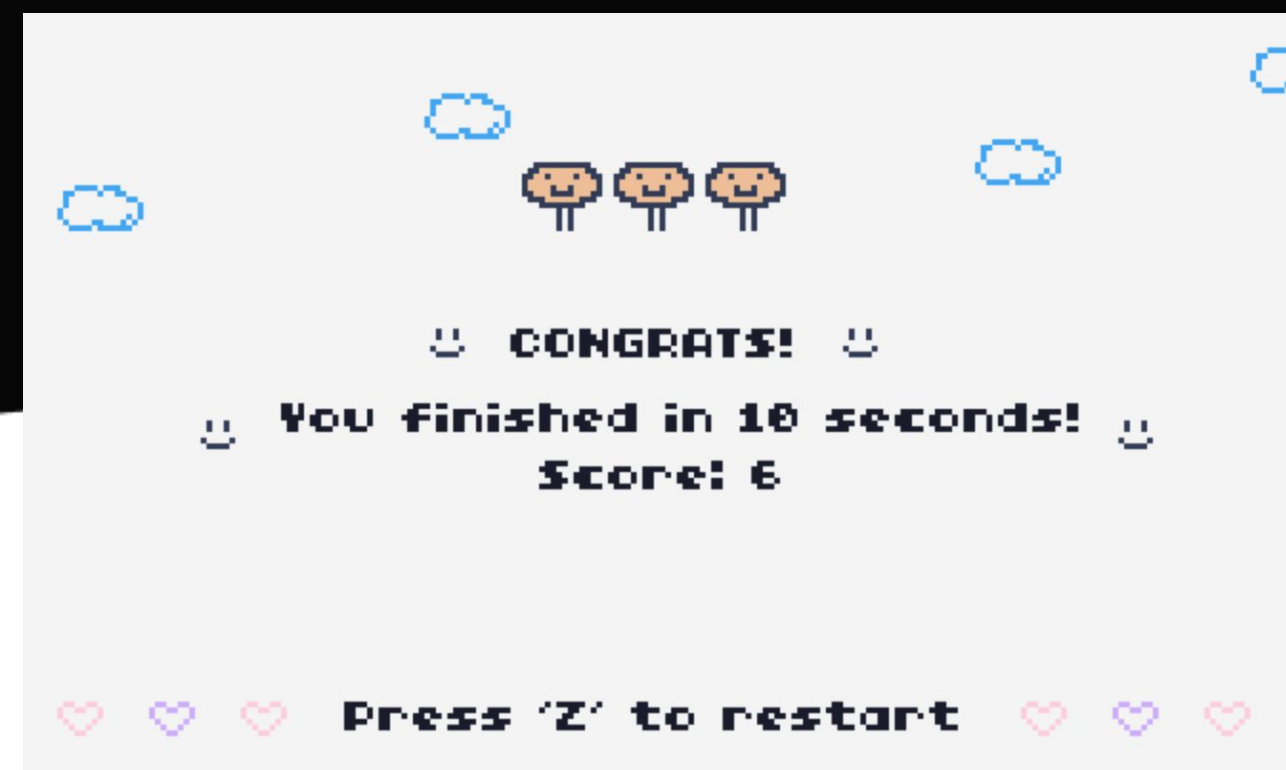
Obstacles: move up/down or left/right continuously across the screen

Collision

Check if distance between two objects are < 120 pixels squared

Player and Target: increase score by 1, target is positioned off-screen

Player and Obstacle: game over



Middle Class + Flyweight



Middle Class

- OOP language for Lua by kikito
- allows for classes, class variables, and objects

Flyweight

- targets and obstacles
- shared data:
 - prototype table (class)
 - ex. methods (movement, etc)
- unshared data:
 - object table

```
local class = require 'middleclass'
```

```
-- bean (target) object
```

```
function Bean:initialize(x, y, vx, vy)
```

```
    self.v = Vector2:new(x, y)
```

```
    self.num = 0
```

```
    self.vx = vx
```

```
    self.vy = vy
```

```
end
```

```
-- move target
```

```
function Bean:move()
```

```
    self.v.x = self.v.x + self.vx
```

```
    self.v.y = self.v.y + self.vy
```

```
end
```

Object Pool



- make targets and obstacles **once** as arrays and reuse throughout game
- position targets off screen after collision
- reposition to starting location at beginning of each round



```
-- reset location of targets
function resetBeans()
  for i=1,#beans do
    if i<= 3 then
      beans[i].v.x = 80+(i*40)
      beans[i].v.y = 20
    else
      beans[i].v.x = -40+(i*40)
      beans[i].v.y = 80
    end
    beans[i]:drawBeans()
  end
end

-- do same for obstacles
```

Game State



- changes 'currState' to decide what to perform next, using a gameState table
- makes our game clearer and easier to manage

```
local gameState = {}  
local currState = "start"
```

```
function gameState:setState(state)  
    currState = state  
end
```



```
function gameState:start()  
    ...  
    --start game when 'Z' pressed  
    if btn(4) then  
        ...  
        self:setState("play")  
    end  
end
```



```
function gameState:play()  
    if bird.gameOver == 1 then  
        self:setState("over")  
    else  
        -- display cumulative score  
        ...  
        --target logic  
        ...  
        --obstacle logic  
        ...  
    end  
end
```



```
function gameState:over()  
    endGame()  
    if btn(4) then  
        restartGame()  
        self:setState("start")  
    end  
end
```

Game Loop



- runs continuously throughout game play
 - process user input without blocking
 - increments frame count every loop to control game speed
- allows for use on different hardware



Our game loop: **function TIC()**

```
function TIC()
  cls(13)
  map(30,0,30,17,0,0)

  if t==1 then
    makeBeans()
    makeHeads()
  end

  gameState:run()

  s=s+1
end
```

Game



Move the Bird to eat the Jellybeans!

Avoid hitting the floating heads.



Press 'Z' to start

[Click to Play](#)





Thank you

Any Questions?