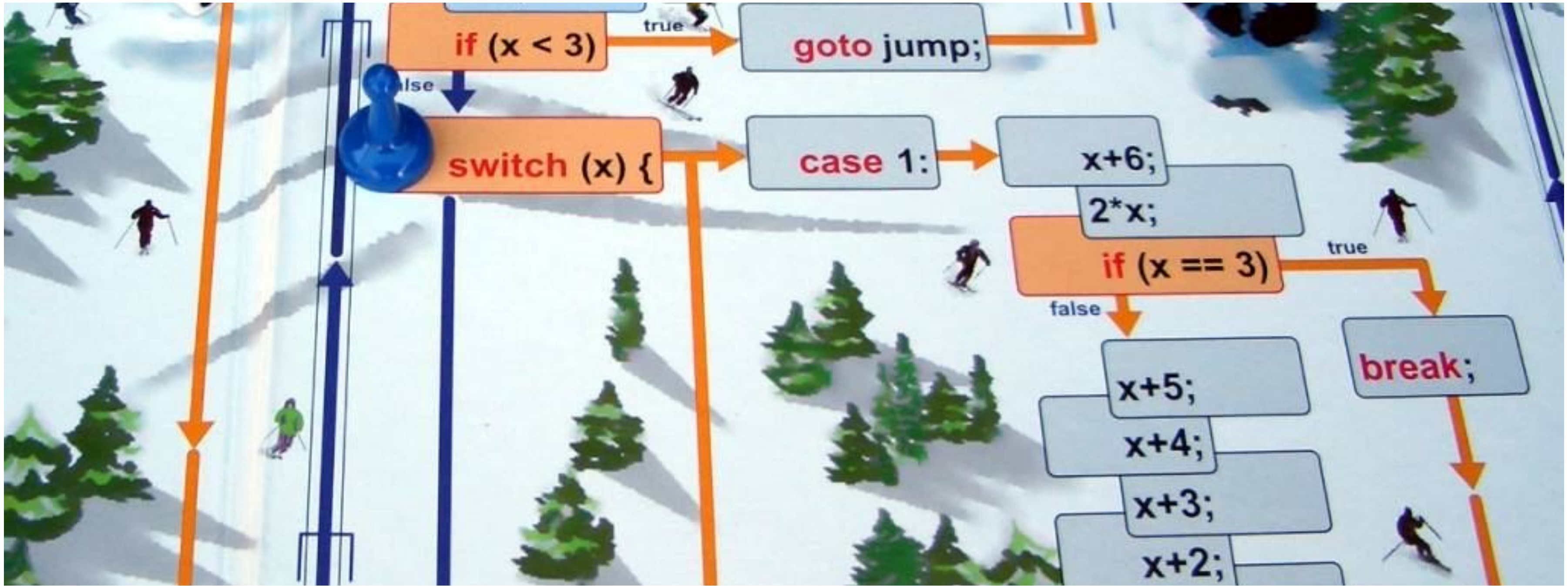


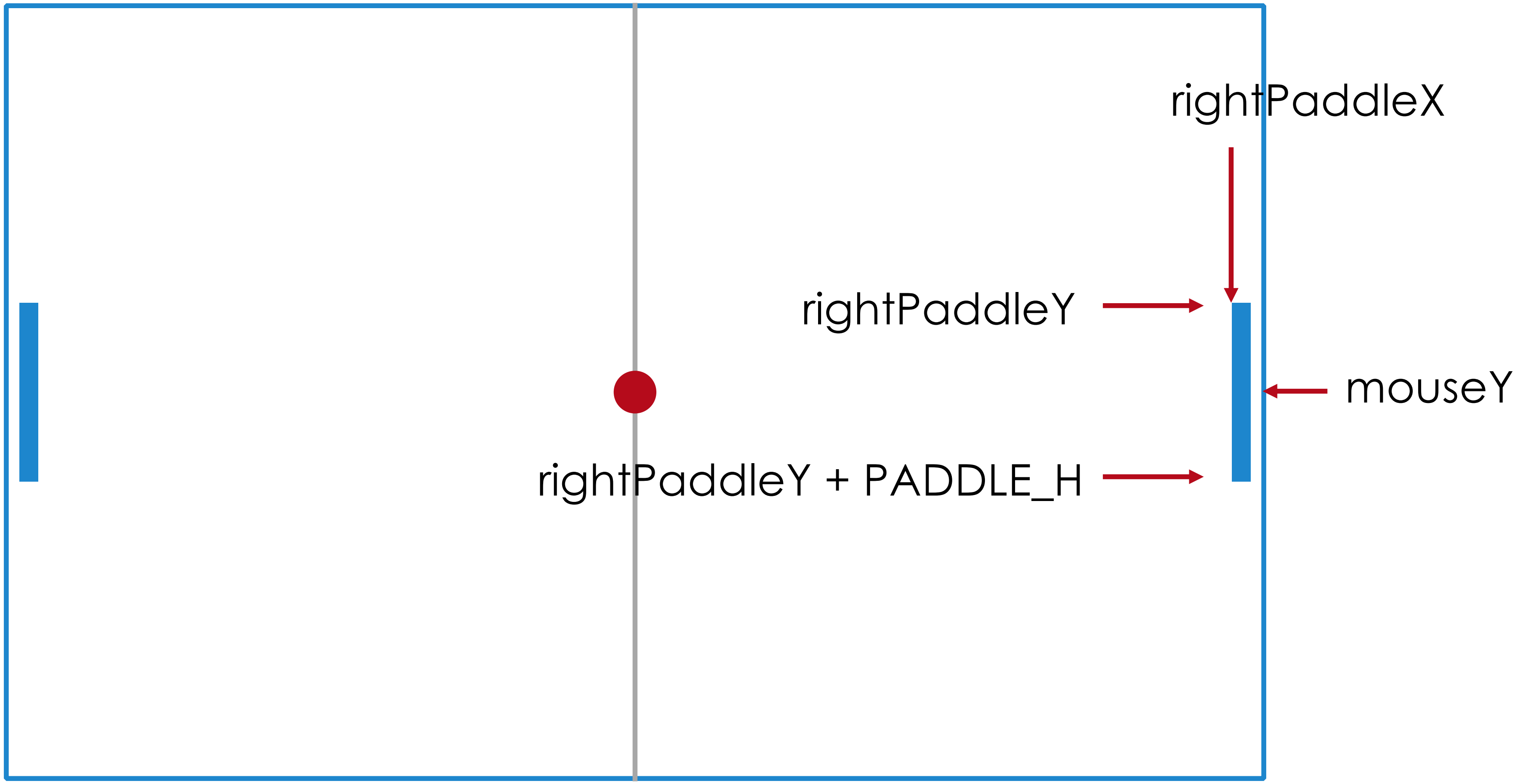
Creative Coding 2023

Instructor: Neng-Hao (Jones) Yu
Course website: https://openprocessing.org/class/83620

Exercise

- ❑ Press any key to pause the game
- ❑ ~~Use mouse to control vertical position of the paddle~~
- ❑ Hit detection: bounce the ball if it hits the paddle
- ❑ Add an unbeatable AI:
  left paddle will always follow the ball’s y position
- ❑ Print scores and remaining lives
  - ◘ Initial score = 0; add 10 points per hit
  - ◘ Initial lives = 3; lost a life for not hitting the ball
  - ◘ Print “Game over” when the player loses all lives

Starter code: https://openprocessing.org/sketch/1871636



rightPaddleX
rightPaddleY
mouseY
rightPaddleY + PADDLE_H

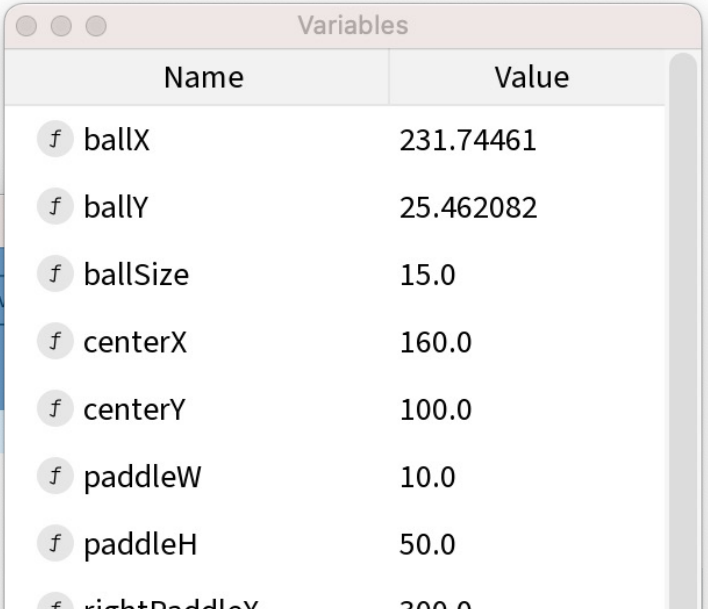
Types of bugs

- Compile-time errors
  - syntax error, type error…
- Runtime exceptions
  - dividing by zero, null pointer…
- Logic flaws
  - the program does not behave as intended

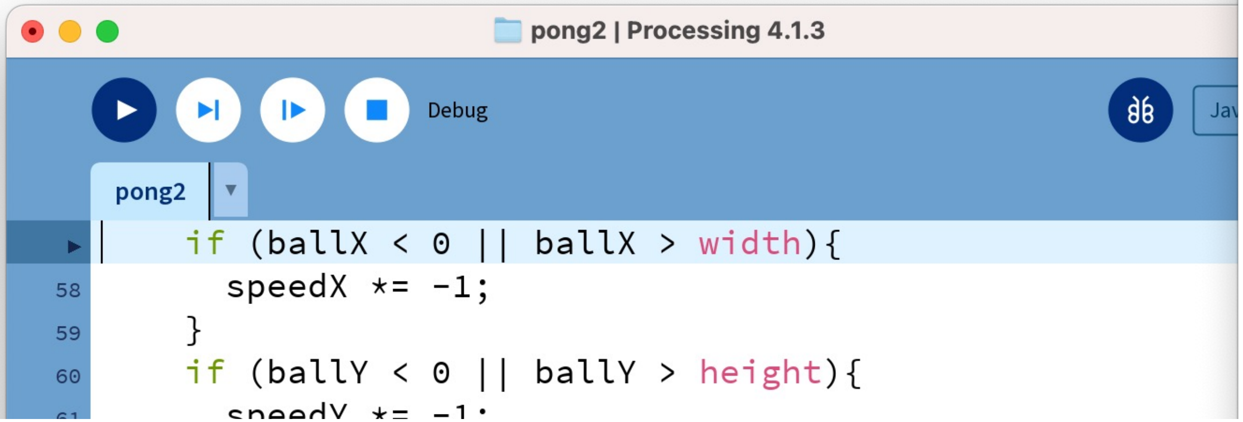
http://stackoverflow.com

Debugging process

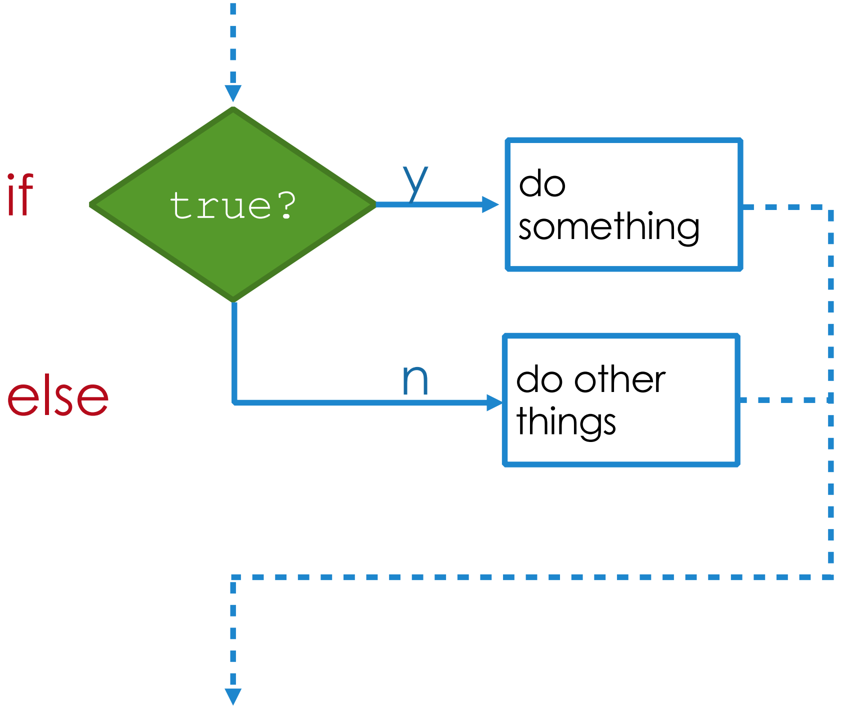
- Break down your problem into smaller pieces
  - Use // or /* */ to temporarily disable parts of your code
- Use println() to check variable values in the Console window
- Use debugger tool



Name	Value
ballX	231.74461
ballY	25.462082
ballSize	15.0
centerX	160.0
centerY	100.0
paddleW	10.0
paddleH	50.0
rightPaddleX	300.0



```
    if (ballX < 0 || ballX > width){
      speedX *= -1;
    }
    if (ballY < 0 || ballY > height){
      speedY *= -1;
```



if

true?

y

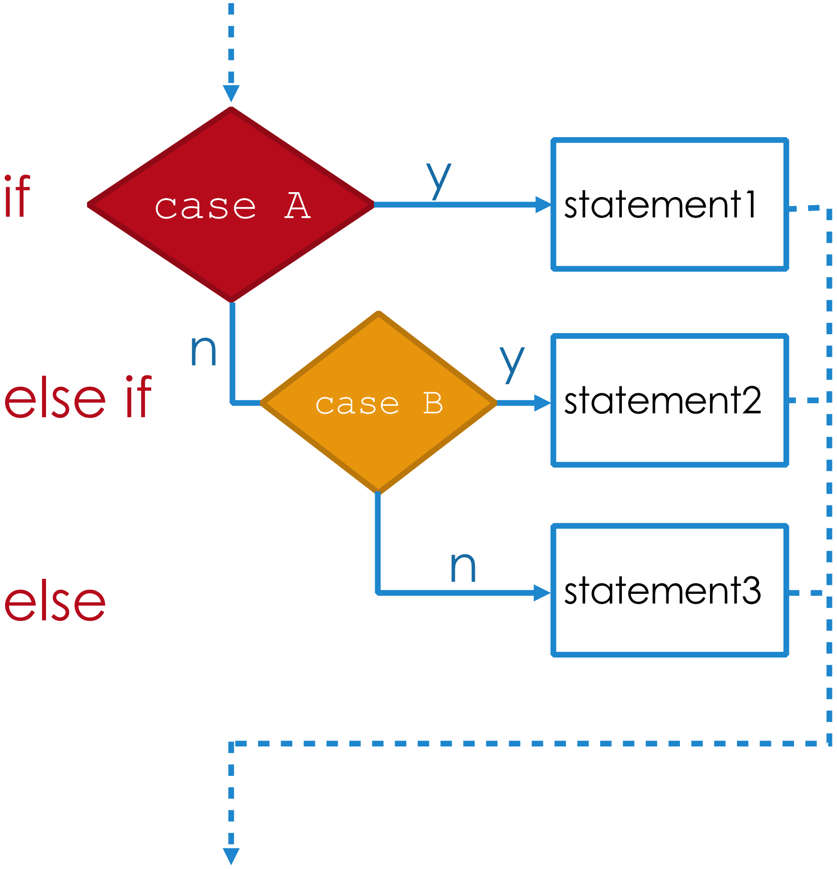
do something

else

n

do other things

```
if (expression) {

} else {

}
```



if

case A

y

statement1

n

else if

case B

y

statement2

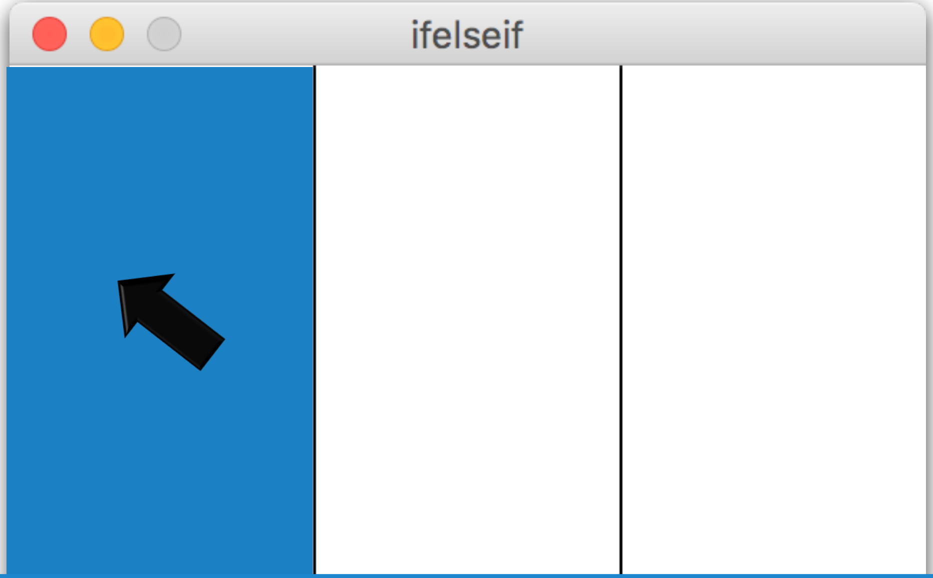
else

n

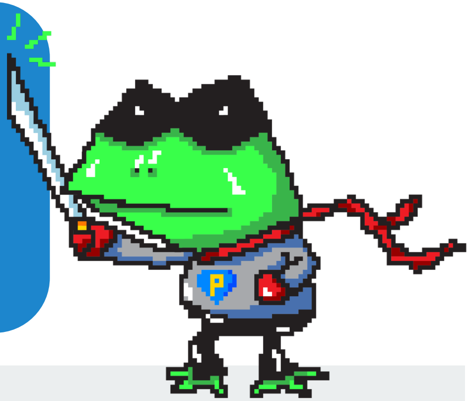
statement3

```
if (case_A) {

} else if (case_B) {

} else {

}
```

Highlight mouse-inside area

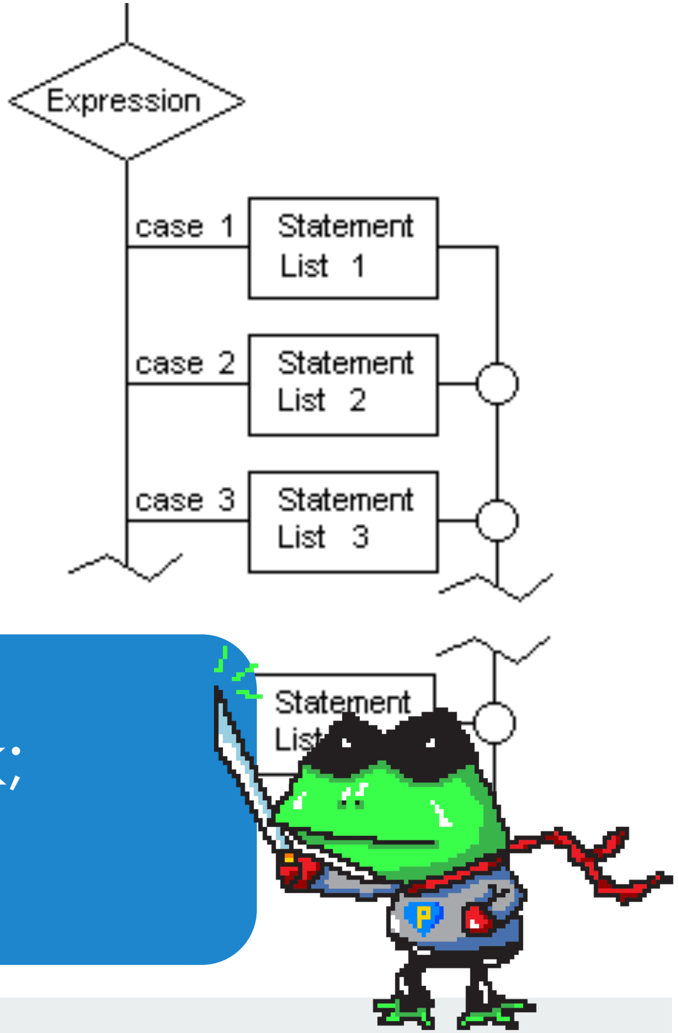


The order of conditions can result in different outcomes



switch statements

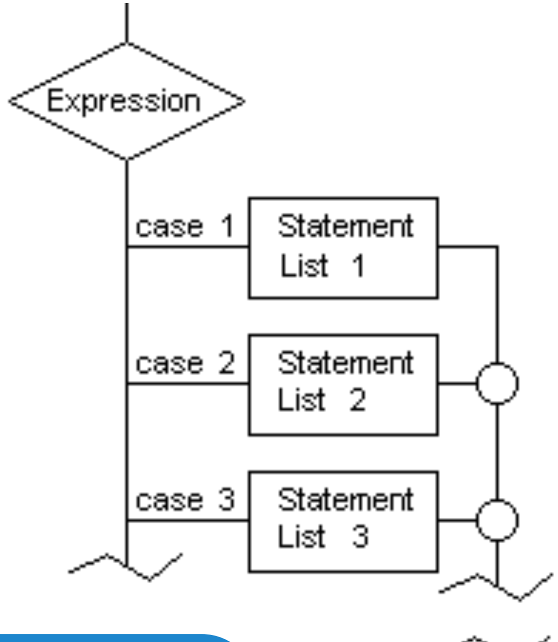
```
                   int or char
        switch ( expression ) {
              case cond1:
if                      do something…;
                        break;
else if       case cond2:
                        do something…;
                        break;
else          default:
```



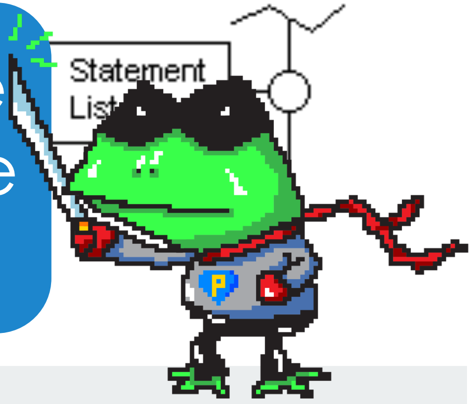
Don’t forget to add break; at the end of each case

switch statements

```
switch ( expression ) {
        case cond1:
                do something…;
                ~~break;~~
        case cond2:
                do something…;
                break;
        default:
```



W/o the break, the code will continue to execute the next case(s) even if the condition(s) are not met



```
char grade = 'B';
  switch (grade){
    case 'A':
A     println("Great job - you are getting an A");
      break;
    case 'B':
B     println("good job - you are getting a B");
      break;
    case 'C':
C     println("average - you are getting a C");
      break;
    case 'D':
D     println("work harder - you are getting a D");
      break;
    case 'F':
F     println("I'm sorry - you are failing");
      break;
    default:
      println("Invalid data");
      break;
 }
```

Win or lose

```
int rnd;
rnd = (int)random(6)+1;
println(rnd);
switch (rnd){
  case 1:    case 2:    case 3:
      println("win");
      break;
  default:
      println("lose");
}
```

Keyboard control

```
void keyPressed() {
  if (key == CODED) {
      switch( keyCode )
      {
        case UP:
          y -= speed;
          break;

        case DOWN:
          y += speed;;
          break;
      }
  }
}
```

```
void keyPressed() {
  if (keyCode == ENTER) {
   // Enter was pressed
  }else if (key == ’a’){
   // key ’a’ was pressed
  }else if (key == CODED){
   // use switch case here
  }
}
```

https://processing.org/reference/keyCode.html

UFO v1: https://openprocessing.org/sketch/1871773

UFO v2 (smooth): https://openprocessing.org/sketch/1871774

Thrust and decay: https://openprocessing.org/sketch/1871786

State machine

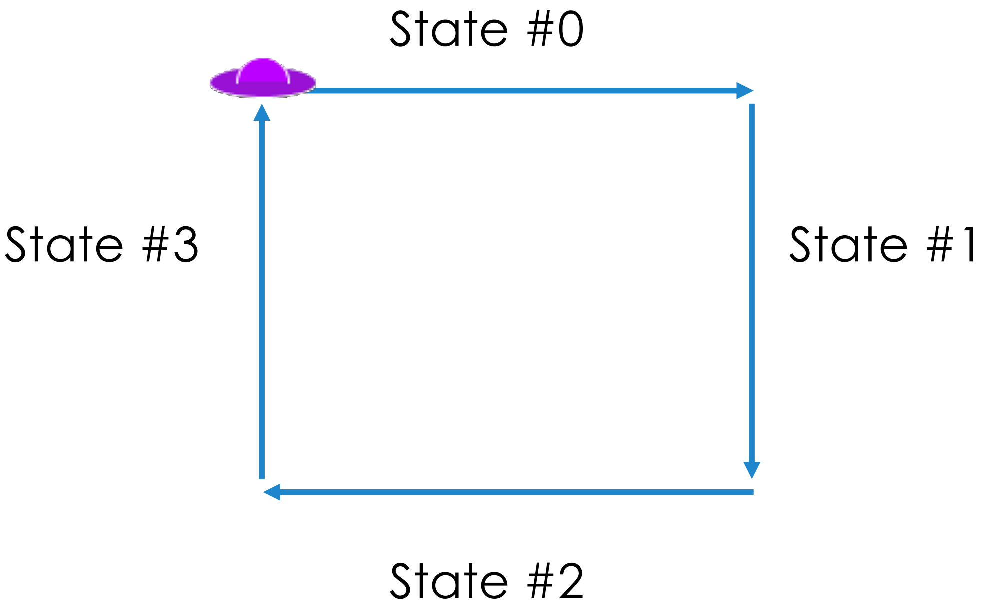
```
final int GO_RIGHT = 0;
final int GO_DOWN = 1;
final int GO_LEFT = 2;
final int GO_UP = 3;


switch (state)
```

state #0: left to right

state #1: top to bottom

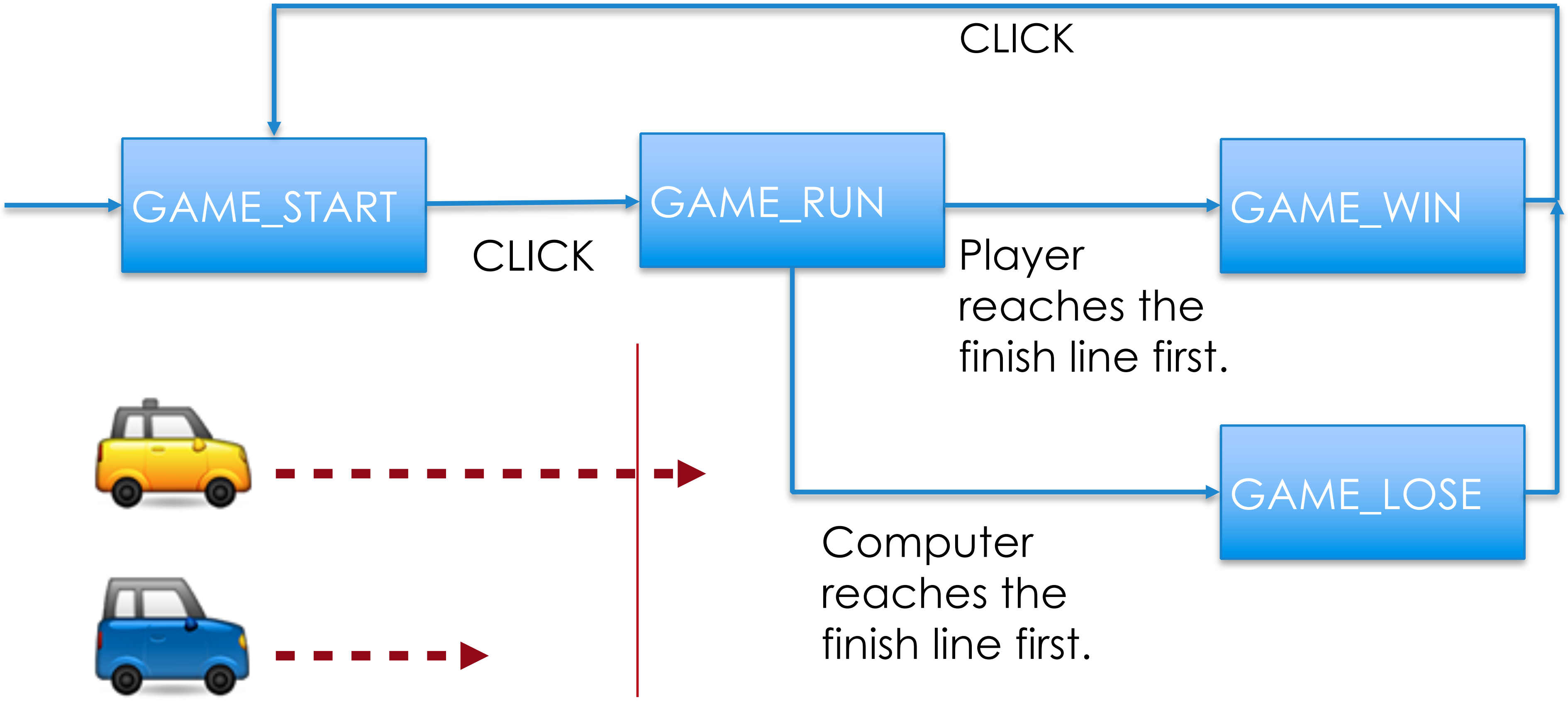
state #2: right to left

state #3: bottom to top



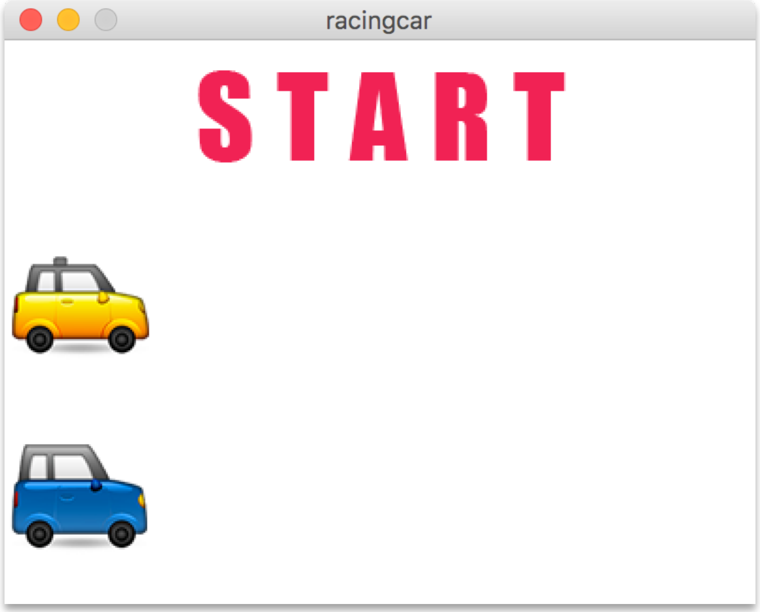
https://openprocessing.org/sketch/1871789

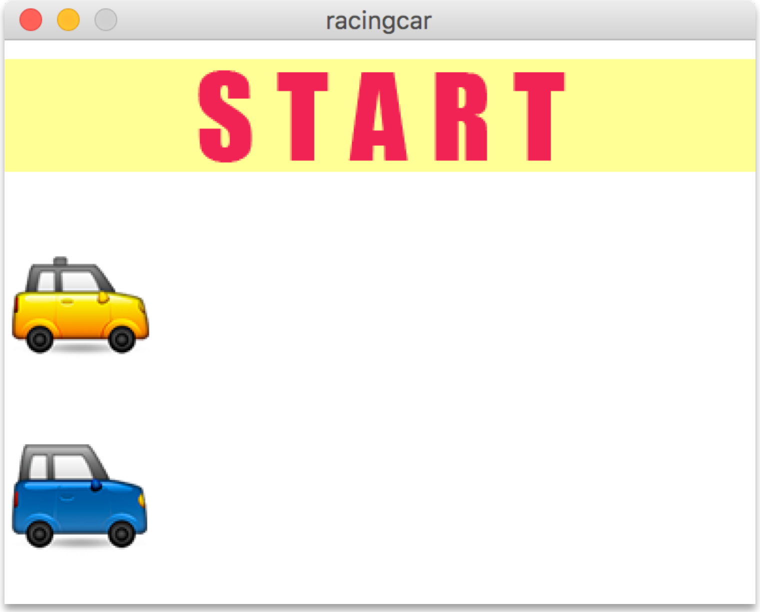
Game states


CLICK
GAME_START
CLICK
GAME_RUN
GAME_WIN
Player reaches the finish line first.
GAME_LOSE
Computer reaches the finish line first.

https://openprocessing.org/sketch/1871827

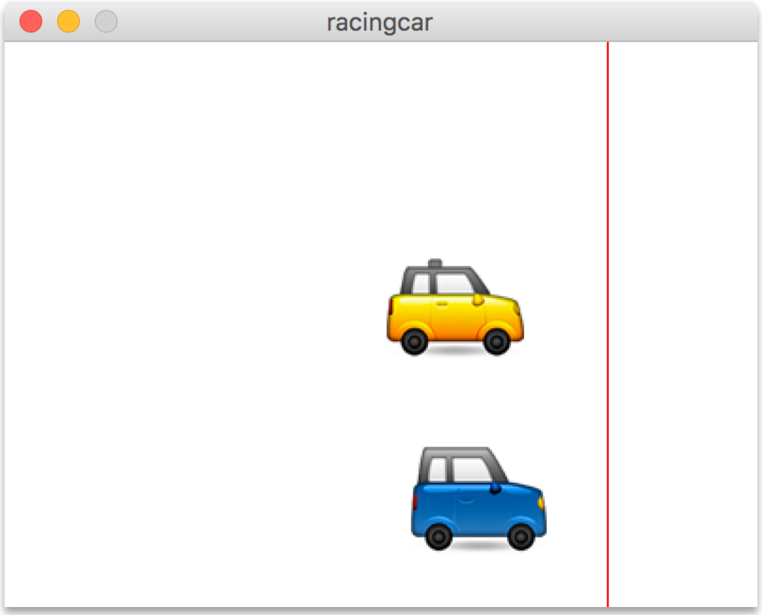
GAME_START





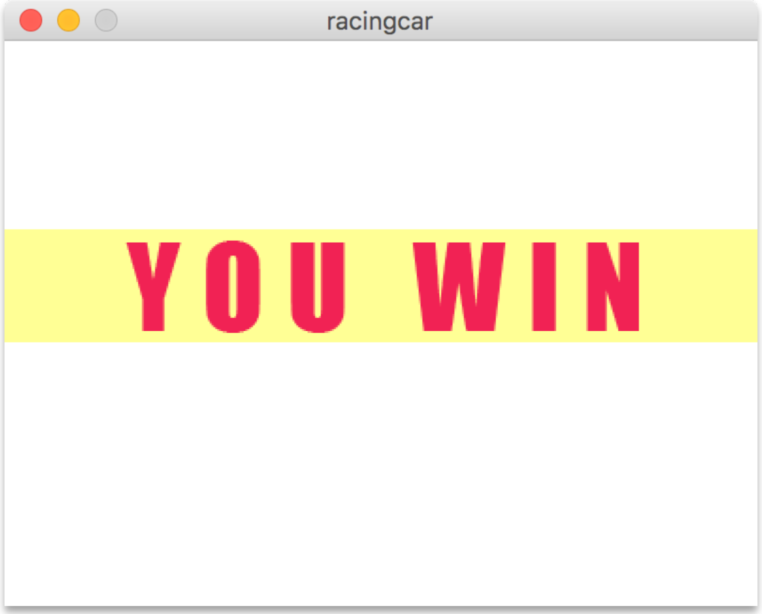
1. Initialize vehicle position
2. Click START to begin the game (go to GAME_RUN)

GAME_RUN

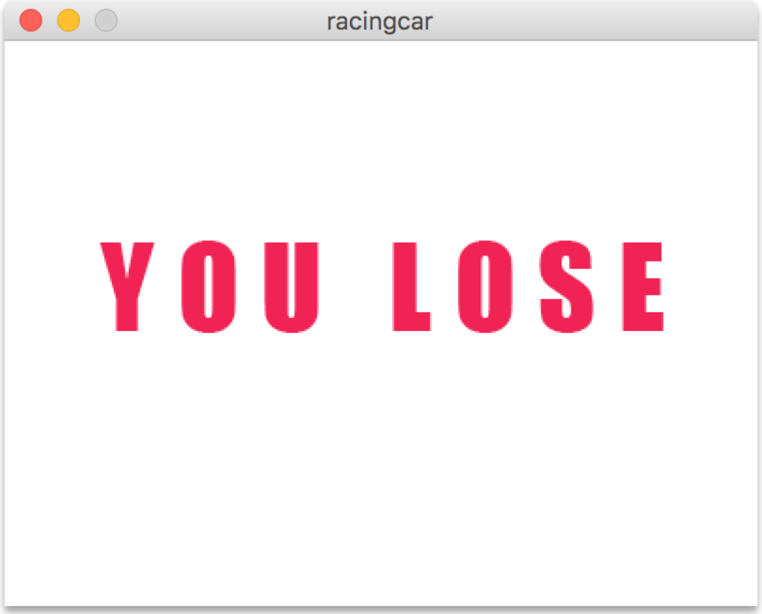


1. Move two vehicles
2. Detect who reaches the finish line first
   (go to GAME_WIN or GAME_LOSE)

GAME_WIN or GAME_LOSE



YOU WIN



YOU LOSE

1. Display the win/lose message
2. Click on the text to restart the game (go to GAME_START)

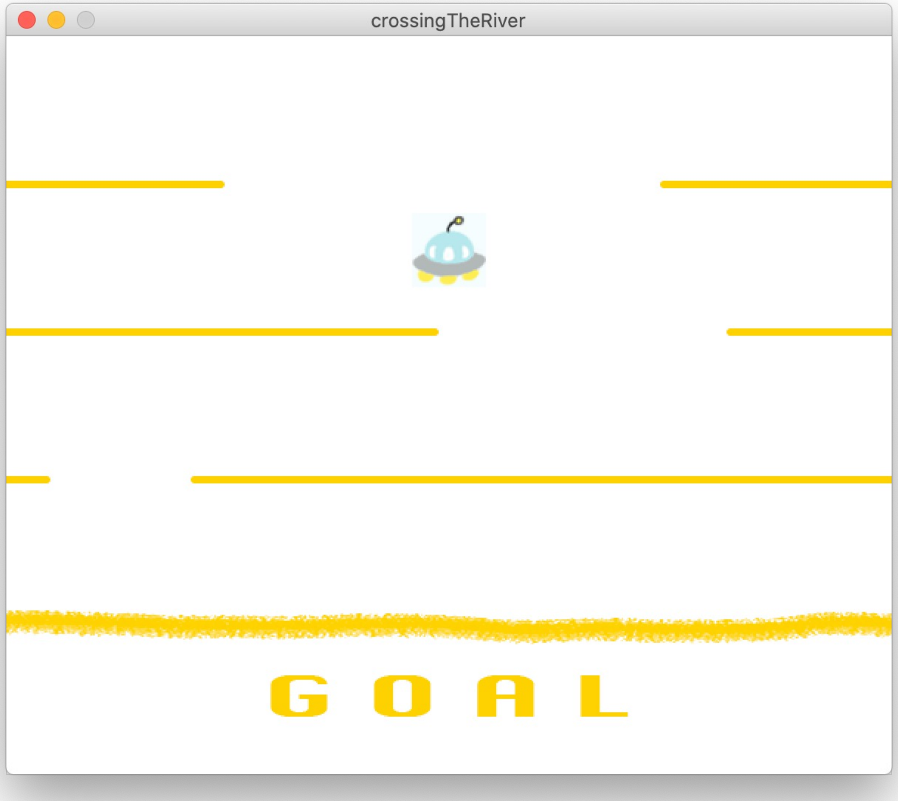
```
final int GAME_START=1, GAME_WIN=2, GAME_LOSE=3, GAME_RUN=4;
int gameState;

void setup(){
  gameState = GAME_START;
}

void draw(){
  switch (gameState){
     case GAME_START:
          // do something
          break;
     case GAME_WIN:
          // do something
          break;
     case GAME_LOSE:
          // do something
          break;
     case GAME_RUN:
          // do something
          break;
  }
}
```

Recap

- Multiple condition judgments
  - if, else if, else
  - switch
- Key control
  - Basic
  - Advance
  - Thrust and decay
- State machine
  - Game flows

飛行船 GO GO

Exercise



Three parts

# 1 Control the spaceship

- Control the spaceship using arrow keys for movement
- The spaceship can only move within the boundaries of the canvas

# 2 Moving walls

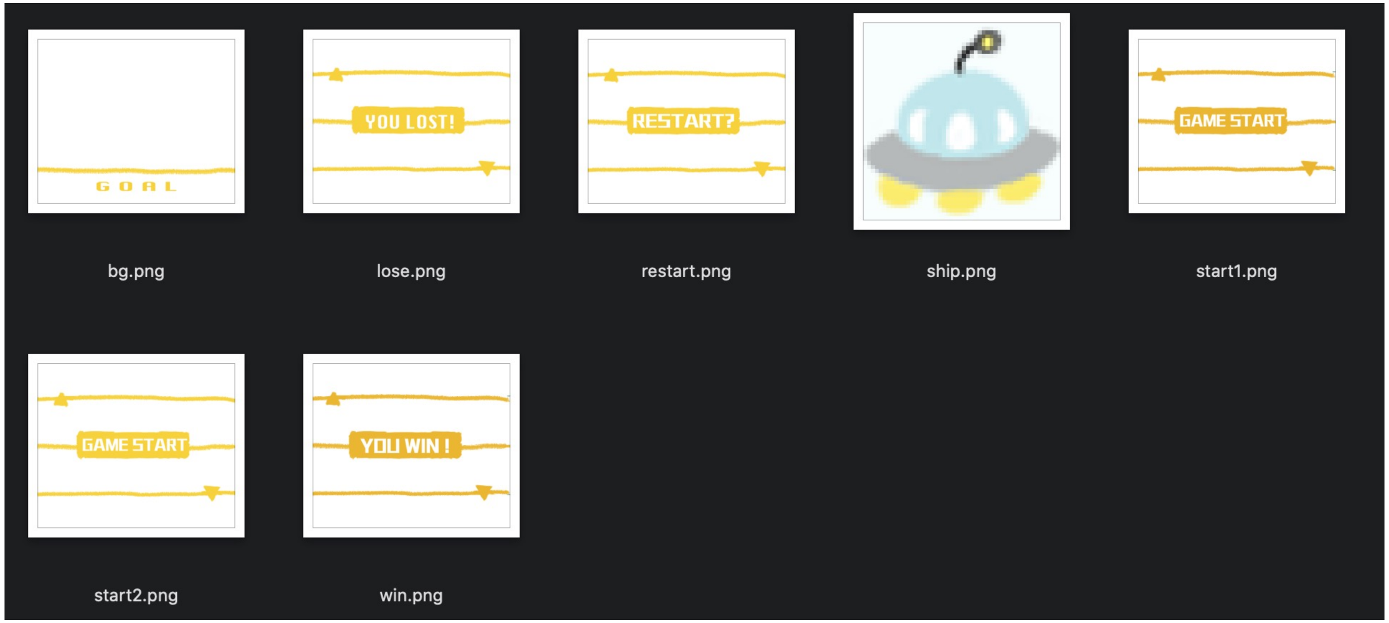
- Set moving walls to obstruct the spaceship's movement

#3 Game flow control

- GAME_START, GAME_RUN, GAME_WIN, GAME_LOSE

Starter code

```
PImage    bg, startNormal, startHover, lose, win, restart, ship;
```



Fork here:
https://classroom.github.com/a/jAqPwh5_

#1 Control the spaceship (15 mins)

Requirements:

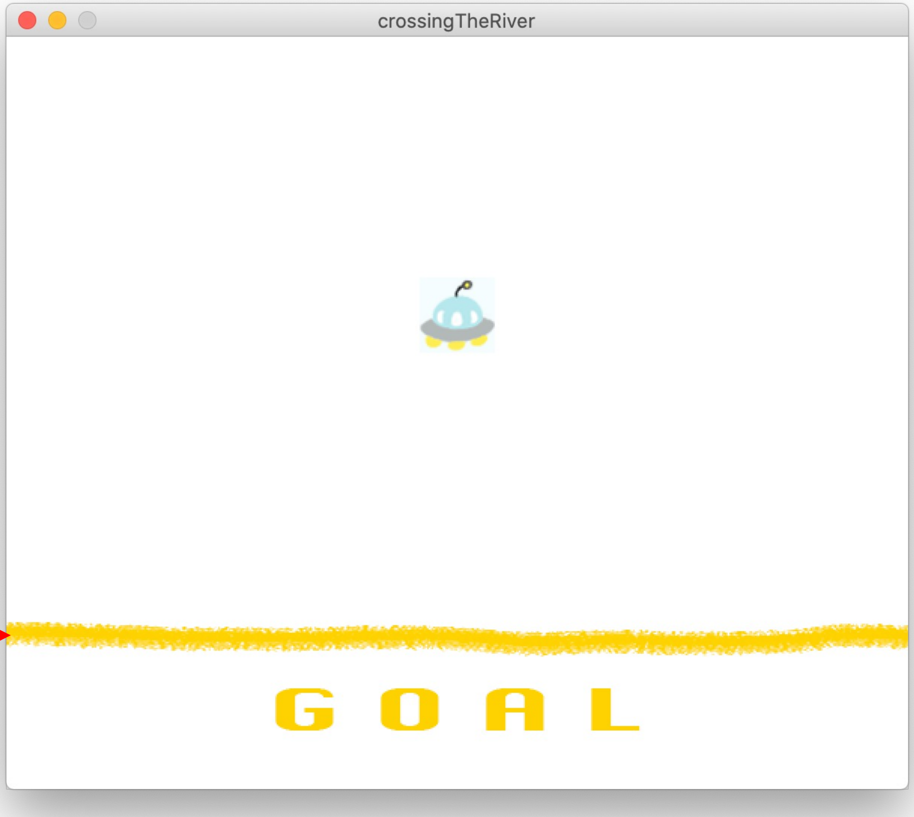
1. Use arrow keys to move the spaceship smoothly
2. The spaceship can only move within the boundaries of the canvas
3. When the spaceship reaches the finish line, display "You win" 。

Initial position: top-center

```
shipX = width / 2 - shipWidth / 2;
shipY = 0;
```

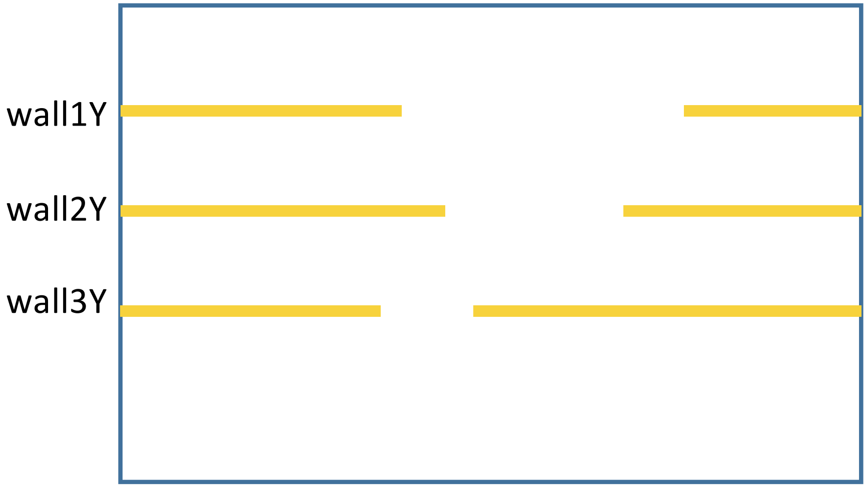
#1 Control the spaceship

When the spaceship crosses the finish line, display "You win".



winningLineY

#2 Moving walls(25 mins)



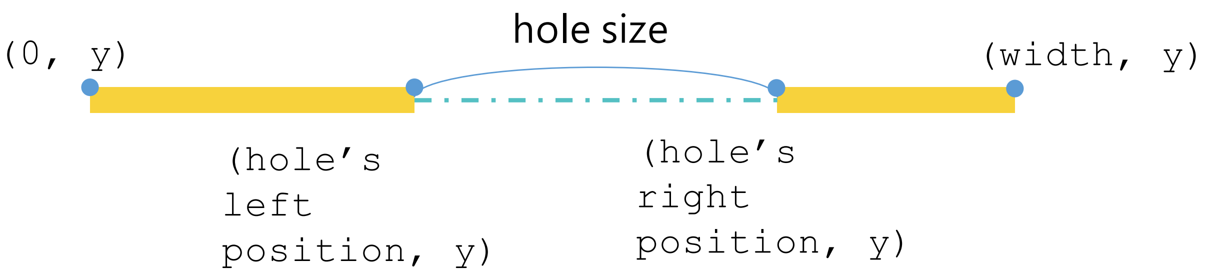
Requirements

1. There are three moving walls with different speeds: 1, 2, and 3 pixels per frame.
2. The opening of each wall will become smaller, with widths of 300, 200, and 100 pixels respectively.
3. If the opening on the right side reaches the right boundary or the opening on the left side reaches the left boundary, reverse the direction of movement.

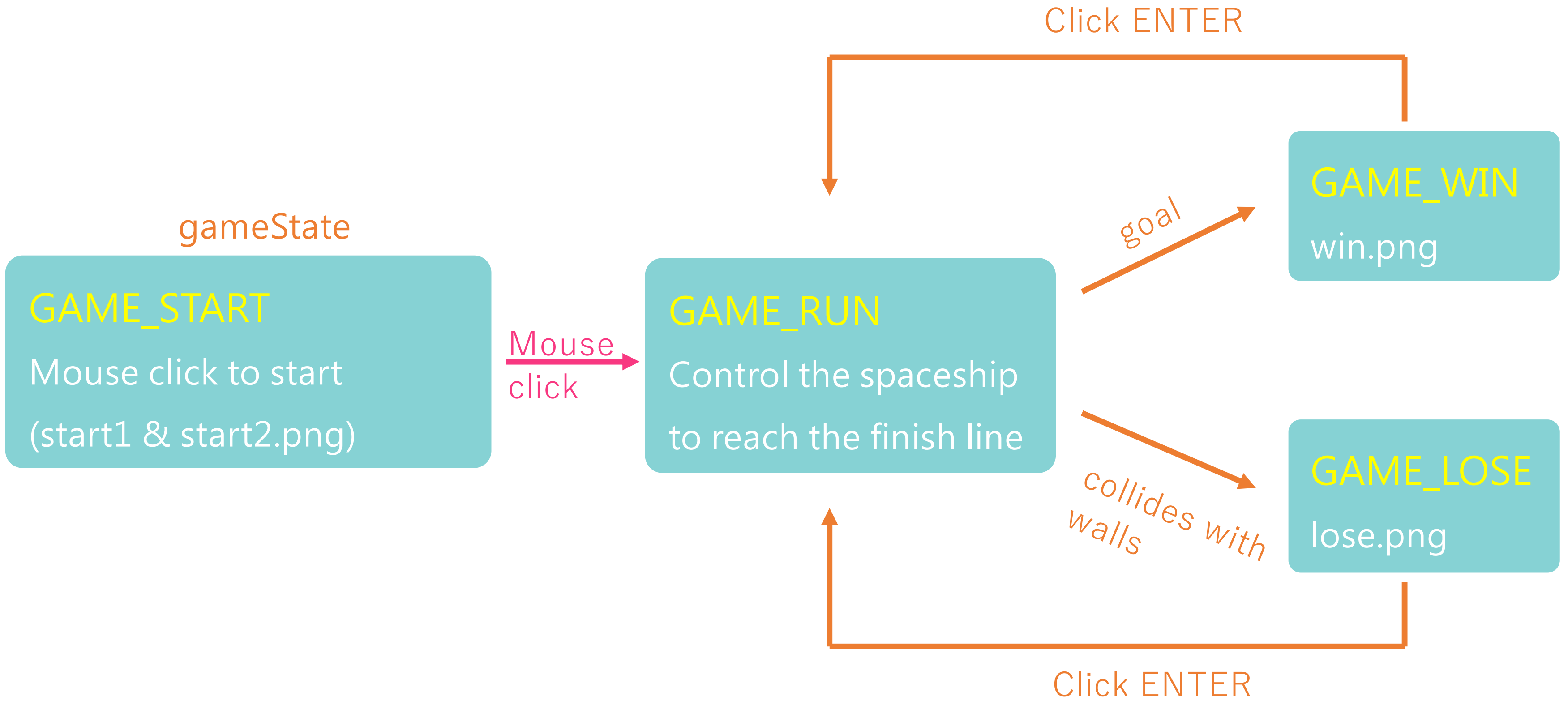
#2 Moving walls

Hint

Wall with a hole in the middle → Divide the wall into left and right sections to draw



#3 Game flow control (20 mins)



gameState

GAME_START

Mouse click to start

(start1 & start2.png)

Mouse click

GAME_RUN

Control the spaceship to reach the finish line

goal

GAME_WIN

win.png

collides with walls

GAME_LOSE

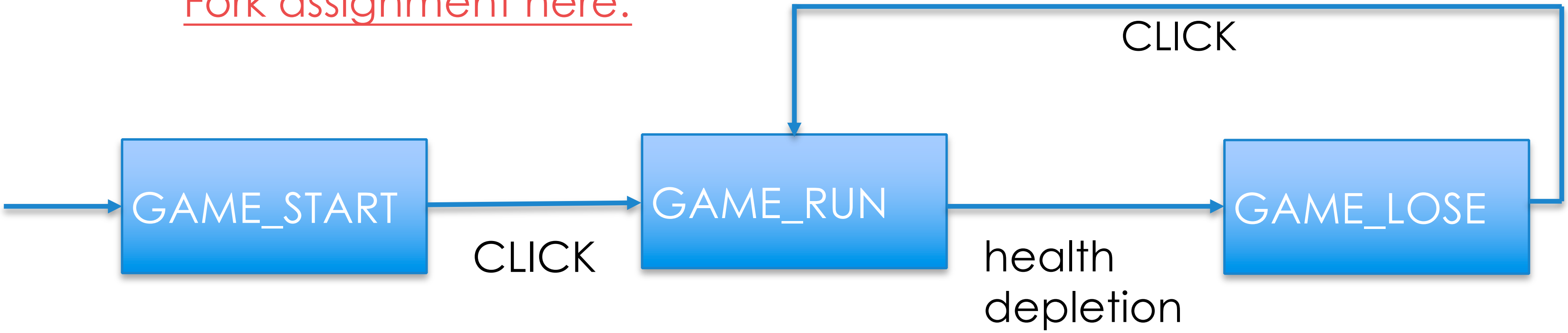
lose.png

Click ENTER

Click ENTER

Assign 2 (Due: 4/4 12pm)

Fork assignment here.


GAME_START

CLICK

GAME_RUN

health depletion

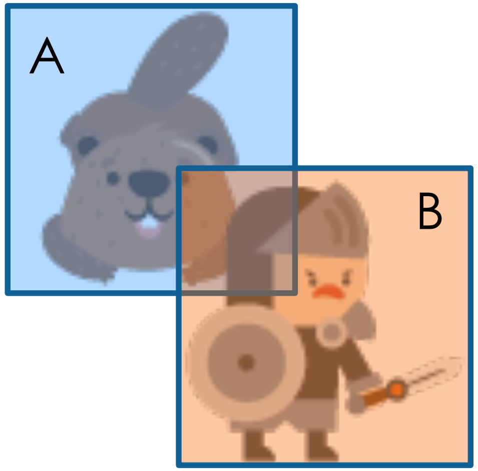
GAME_LOSE

CLICK

Requirements

- 1-star
  - Complete all requirements in assign1
  - Implement the game flow (GAME_START, GAME_RUN, GAME_LOSE), and ensure that each screen functions properly. We will use keys ‘1,2,3’ to check the game state.
  - Player can move smoothly using the arrow keys and cannot move off-screen.
- 2-star
  - Display the health bar with a range of 0-100 and the length of the bar should be proportional to the health value.
  - The player loses 20 health points when hit by an enemy, and gains 10 health points when they collect a treasure.
  - The enemy or treasure will disappear once they collide with the player's spaceship. The enemy will then reappear on the left side of the screen, and the treasure will randomly appear at any point on the screen.
    Hint: use AABB collision detection algorithm
- 3-star
  - The enemy can only fly from left to right, but it will accelerate towards the player when the distance between them is less than 300 pixels.

AABB collision detection

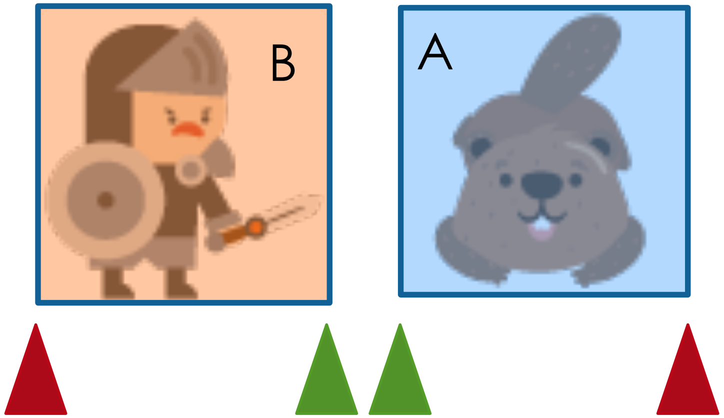


AABB (Axis-Aligned Bounding Boxes):

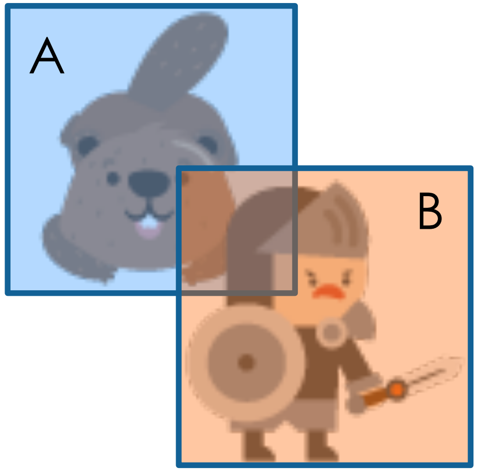
The algorithm checks four conditions

to determine whether the boxes intersect:

- A.left <= B.right ✗
- A.right >= B.left ○
- A.top <= B.bottom
- A.bottom >= B.top



AABB collision detection

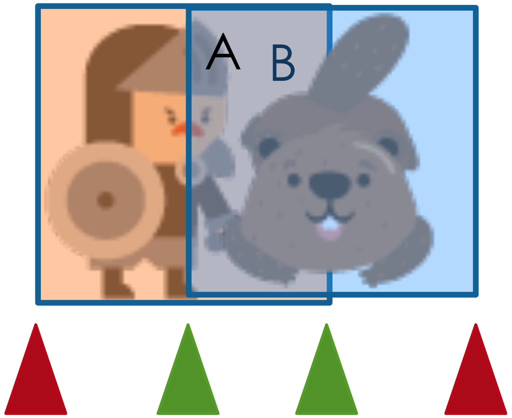


AABB (Axis-Aligned Bounding Boxes):

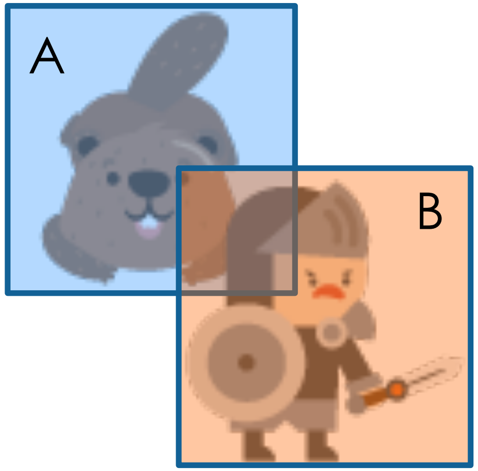
The algorithm checks four conditions

to determine whether the boxes intersect:

- A.left <= B.right ○
- A.right >= B.left ○
- A.top <= B.bottom
- A.bottom >= B.top



AABB collision detection

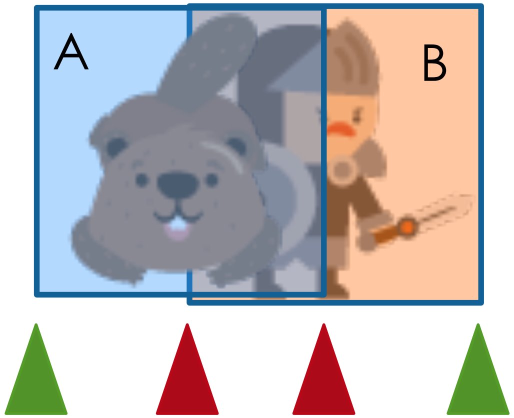


AABB (Axis-Aligned Bounding Boxes):

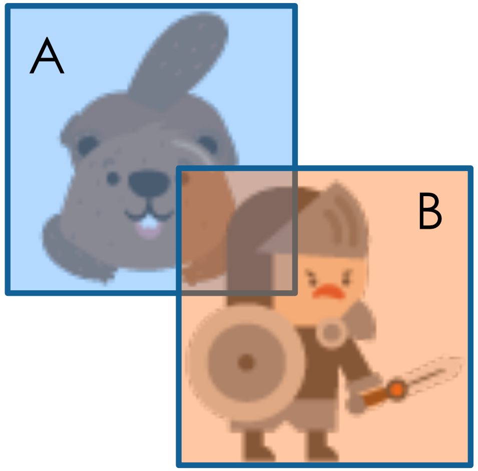
The algorithm checks four conditions

to determine whether the boxes intersect:

- A.left <= B.right ○
- A.right >= B.left ○
- A.top <= B.bottom
- A.bottom >= B.top



AABB collision detection



AABB (Axis-Aligned Bounding Boxes):

The algorithm checks four conditions

to determine whether the boxes intersect:

- A.left <= B.right ○
- A.right >= B.left ✗
- A.top <= B.bottom
- A.bottom >= B.top