20 Minutes to Snake

Reminders & Announcements

- Exam 2 pretty much graded, expect grades early next week
    - As a class, you did well!
- TA Applications due tonight @ 11:59pm
    - Check Ed for link
    - No late days on this :)

HW8 (Snake) Released

- Cannot be dropped!
- **Hard** deadline on Dec 9 @ 11:59pm
  - No late days accepted
  - Personal emergencies -> temporary incomplete grade in the class
  - Start now so you can use the whole time!

Game Loops

Common Structure For A Game

We've built a game! It's *Furious Flying Fish*. We can learn from that experience that it's helpful for a class to have:

- a *static* game class with a main loop, e.g. **FuriousFish.java**
- a "game entity collection" class, e.g. **Arena.java**
- classes for individual game entities, e.g. **Fish.java** and **Target.java**

A Generic Approach

`main` is a method inside of our executable class called `Game.java`.

- initializes the game's components
- defines the animation loop
- repeatedly asks the game's components to *update* and then *draw*

Our "entity collection" class is called `Board.java`

A Generic Approach to Main

```
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board(); // ⬅⬅⬅⬅⬅⬅

    while (gameBoard.isRunning()) {
        gameBoard.update();
        gameBoard.draw();
    }

    gameBoard.displayGameOver();
}
```

How many times is this line of code executed?

A Generic Approach to Main

```java
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board(); // ⬅⬅⬅⬅⬅⬅

    while (gameBoard.isRunning()) {
        gameBoard.update();
        gameBoard.draw();
    }

    gameBoard.displayGameOver();
}
```

How many times is this line of code executed? **ONCE!**

A Generic Approach to Main

```java
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board(); // ⬅⬅⬅⬅⬅⬅

    while (gameBoard.isRunning()) {
        gameBoard.update();
        gameBoard.draw();
    }

    gameBoard.displayGameOver();
}
```

What conditions must be met for this line to be executed?

A Generic Approach to Main

```
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board(); // ⬅⬅⬅⬅⬅⬅

    while (gameBoard.isRunning()) {
        gameBoard.update();
        gameBoard.draw();
    }

    gameBoard.displayGameOver();
}
```

What conditions must be met for this line to be executed?

NONE, ALWAYS HAPPENS AT THE START.

A Generic Approach to Main

```java
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board();

    while (gameBoard.isRunning()) {
        gameBoard.update(); // ⬅️⬅️⬅️⬅️⬅️⬅️
        gameBoard.draw();
    }

    gameBoard.displayGameOver();
}
```

How many times is this line executed?

A Generic Approach to Main

```
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board();

    while (gameBoard.isRunning()) {
        gameBoard.update(); // ⬅⬅⬅⬅⬅⬅
        gameBoard.draw();
    }

    gameBoard.displayGameOver();
}
```

How many times is this line executed? **MANY MANY TIMES**

A Generic Approach to Main

```java
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board();

    while (gameBoard.isRunning()) {
        gameBoard.update(); // ⬅⬅⬅⬅⬅⬅
        gameBoard.draw();
    }

    gameBoard.displayGameOver();
}
```

What conditions must be met for this line to be executed?

A Generic Approach to Main

```java
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board();

    while (gameBoard.isRunning()) {
        gameBoard.update(); // ⬅️⬅️⬅️⬅️⬅️⬅️
        gameBoard.draw();
    }

    gameBoard.displayGameOver();
}
```

What conditions must be met for this line to be executed? **GAME MUST BE RUNNING.**

A Generic Approach to Main

```java
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board();

    while (gameBoard.isRunning()) {
        gameBoard.update();
        gameBoard.draw();
    }

    gameBoard.displayGameOver(); // ⬅⬅⬅⬅⬅⬅
}
```

How many times is this line executed?

A Generic Approach to Main

```java
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board();

    while (gameBoard.isRunning()) {
        gameBoard.update();
        gameBoard.draw();
    }

    gameBoard.displayGameOver(); // ⬅️⬅️⬅️⬅️⬅️⬅️
}
```

How many times is this line executed? **ONCE**

A Generic Approach to Main

```java
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board();

    while (gameBoard.isRunning()) {
        gameBoard.update();
        gameBoard.draw();
    }

    gameBoard.displayGameOver(); // ⬅⬅⬅⬅⬅⬅
}
```

What conditions must be met for this line to be executed?

A Generic Approach to Main

```java
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board();

    while (gameBoard.isRunning()) {
        gameBoard.update();
        gameBoard.draw();
    }

    gameBoard.displayGameOver(); // ⬅⬅⬅⬅⬅⬅
}
```

What conditions must be met for this line to be executed? **GAME MUST HAVE STOPPED RUNNING**

Restarting?

HW8 requirements:

- After the player loses, display a "Game Over" screen.
- While the "Game Over" screen is showing, the user can press a key to restart the game.

This often proves challenging for students.

A Common Mistake. Why?

```
public static void main(String[] args) {

    PennDraw.setCanvasSize(500, 500);
    PennDraw.enableAnimation(30);
    Board gameBoard = new Board();

    while (gameBoard.isRunning()) {
        gameBoard.update();
        gameBoard.draw();
    }

    gameBoard.displayGameOver();
    if (PennDraw.hasNextKeyTyped()) {
        gameBoard.restart();
    }
}
```

Don't Forget Your Basics of Control Flow!

```java
while (gameBoard.isRunning()) {
    gameBoard.update();
    gameBoard.draw();
}
gameBoard.displayGameOver();
if (PennDraw.hasNextKeyTyped()) {
    gameBoard.restart();
}
```

Stuff after the while loop is evaluated exactly one time, so if there's no key pressed at that precise moment, the program just finishes executing!

Other Common Mistakes...

```
while (gameBoard.isRunning()) {
    gameBoard.update();
    gameBoard.draw();
}
gameBoard.displayGameOver();
while (PennDraw.hasNextKeyTyped()) {
    gameBoard.restart();
}
```

Same problem—if the key isn't pressed right after the game is over, the condition is false and the game just ends.

Other Common Mistakes...

```
while (gameBoard.isRunning()) {
    gameBoard.update();
    gameBoard.draw();

    gameBoard.displayGameOver();

    if (PennDraw.hasNextKeyTyped()) {
        gameBoard.restart();
    }
}
```

This does repeatedly check for a key press, but it would draw the "Game Over" and allow a restart while the game is playing.

Think Carefully About Control Flow!

- Remember what it means when a block comes before/after another block.
- Remember what it means when a block is placed inside/outside of another block.
- Remember that you control what boolean expression is used to trigger a conditional/loop!

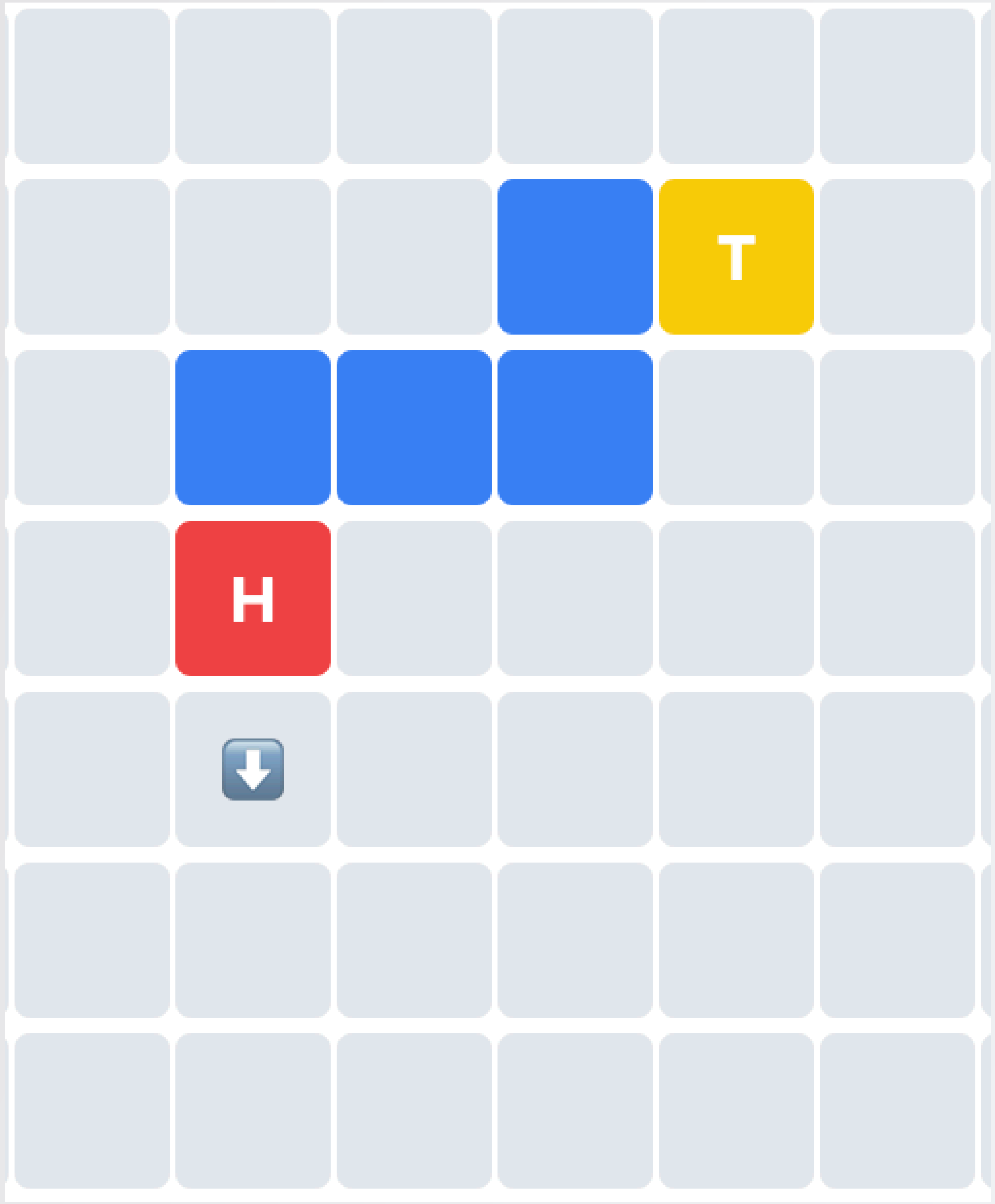
Moving a Snake 🐍

It's often helpful to think of a snake as being made out of a bunch of segments.

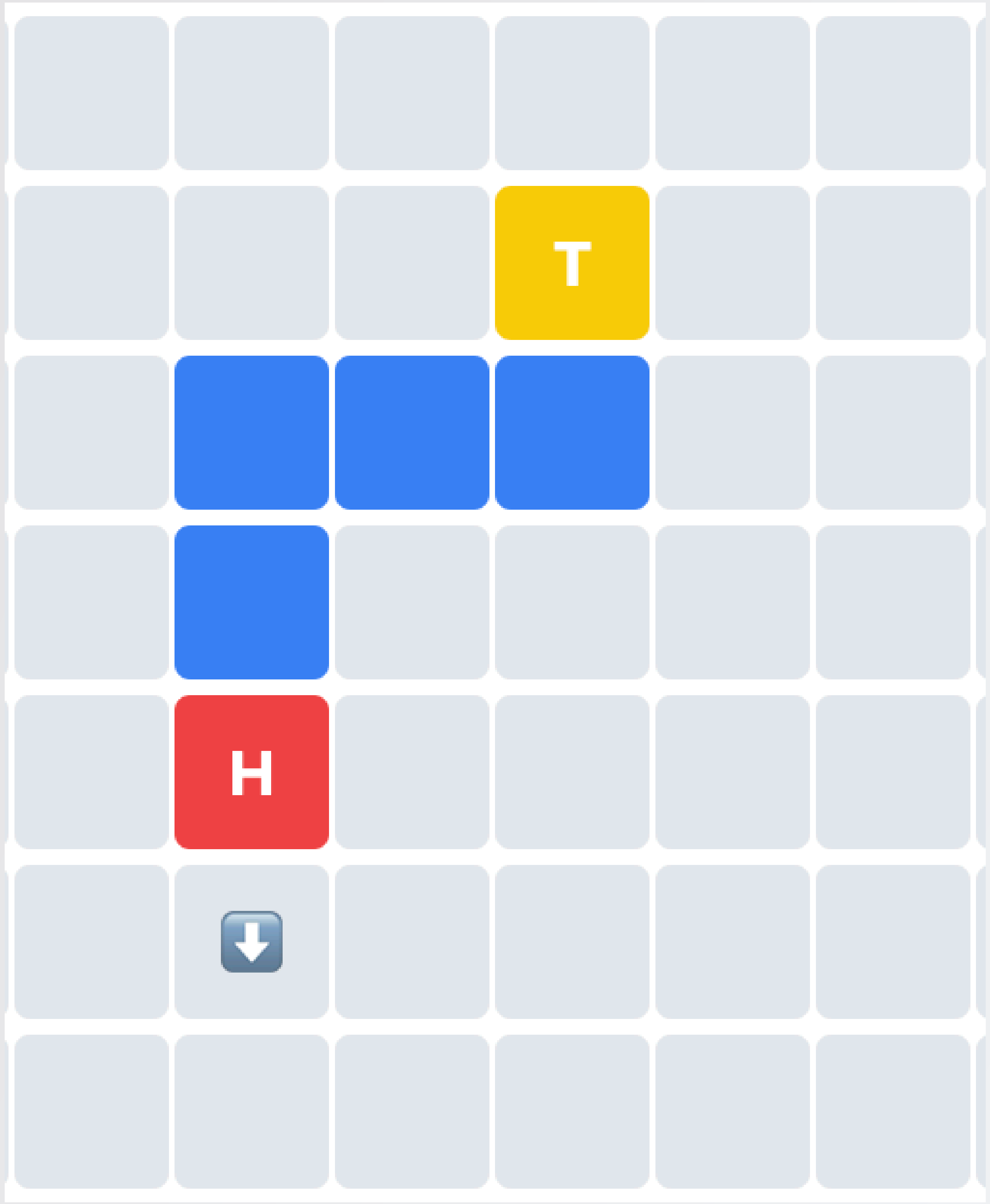
- The **head** is the first segment
- The **tail** is the last segment
- The remaining segments make up the **body**.

S.C.
N.C.
V.
M.
P.
N.J.
N.Y.
N.E.
JOIN, or DIE.

SNAKE



In this picture, the head is red and marked with "H", the tail is yellow and marked with "Y", and the direction of Snake travel is marked with ⬇️.

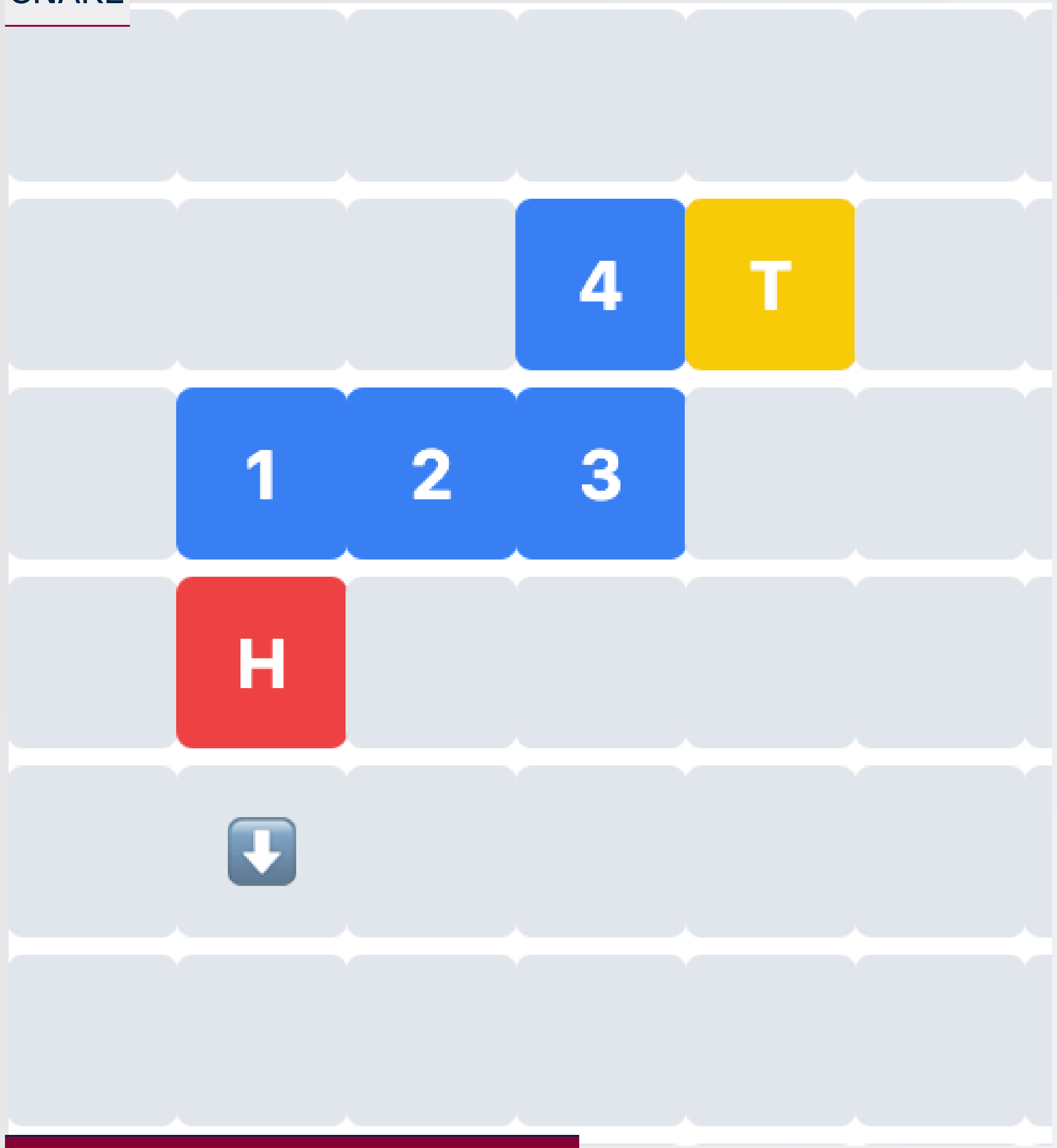


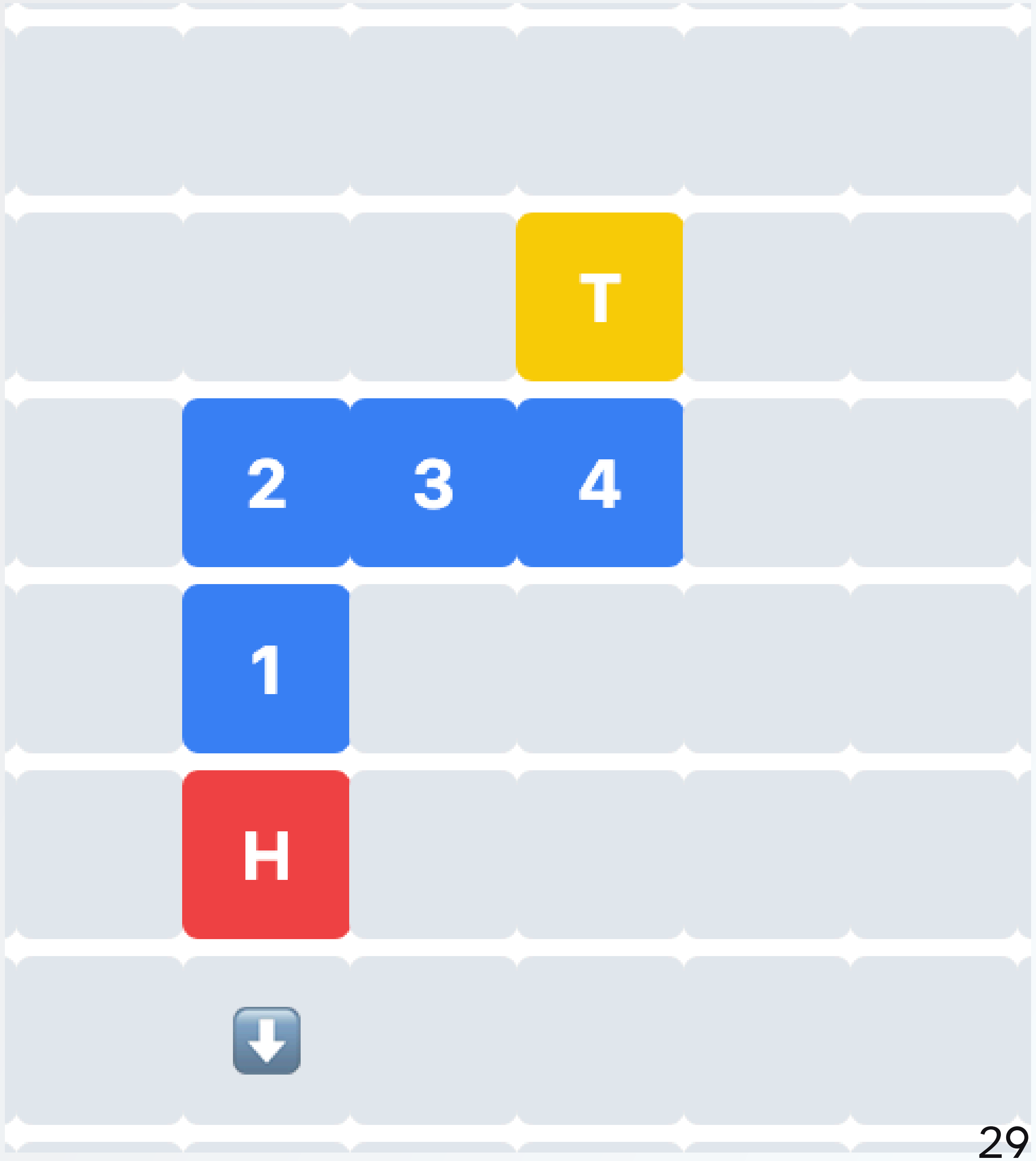
SNAKE

When the Snake moves, starting from the tail, each segment moves into the position of the next segment in line.

The head is slightly different, and moves to the next square in the direction of travel.

SNAKE





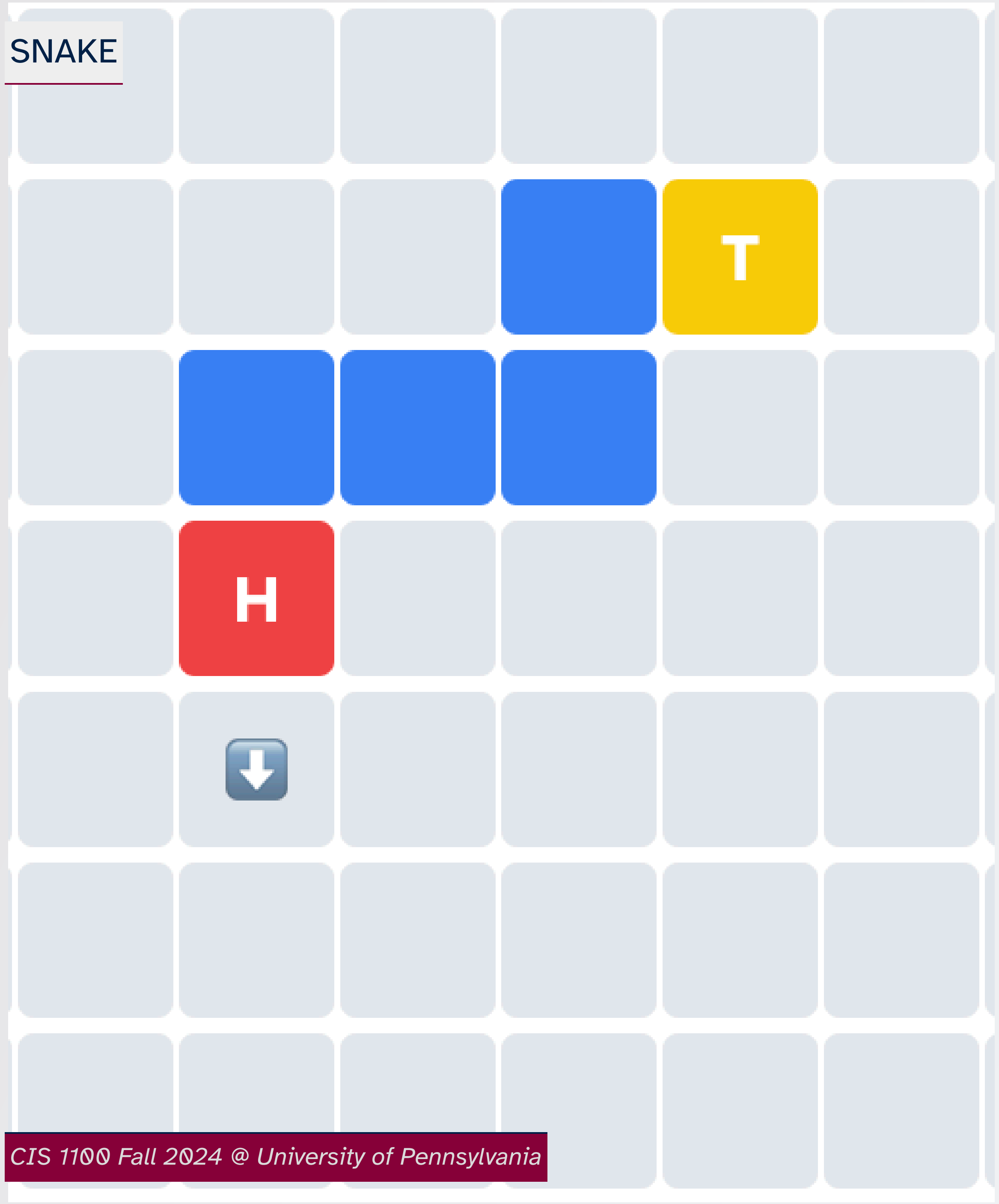
Moving a Snake Isn't Too Complicated…

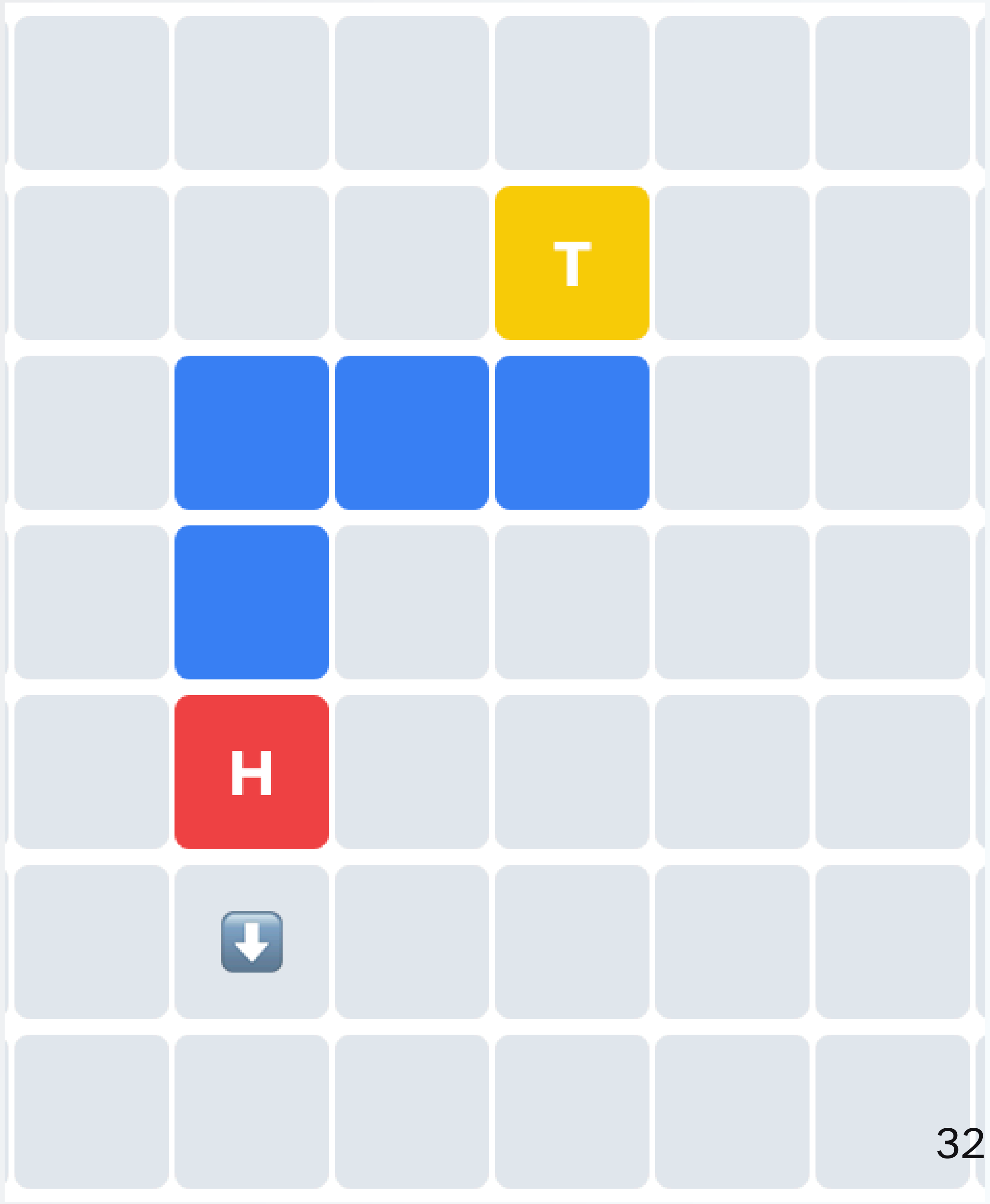
In some messed up pseudocode language I just came up with, we can think of the process of moving a Snake built from a sequence of Segments like so:

```
function moveSnake:

    for (i from n down to 1):
        segments[i].position = segments[i - 1].position

    segments[0].position += directionOfTravel
```

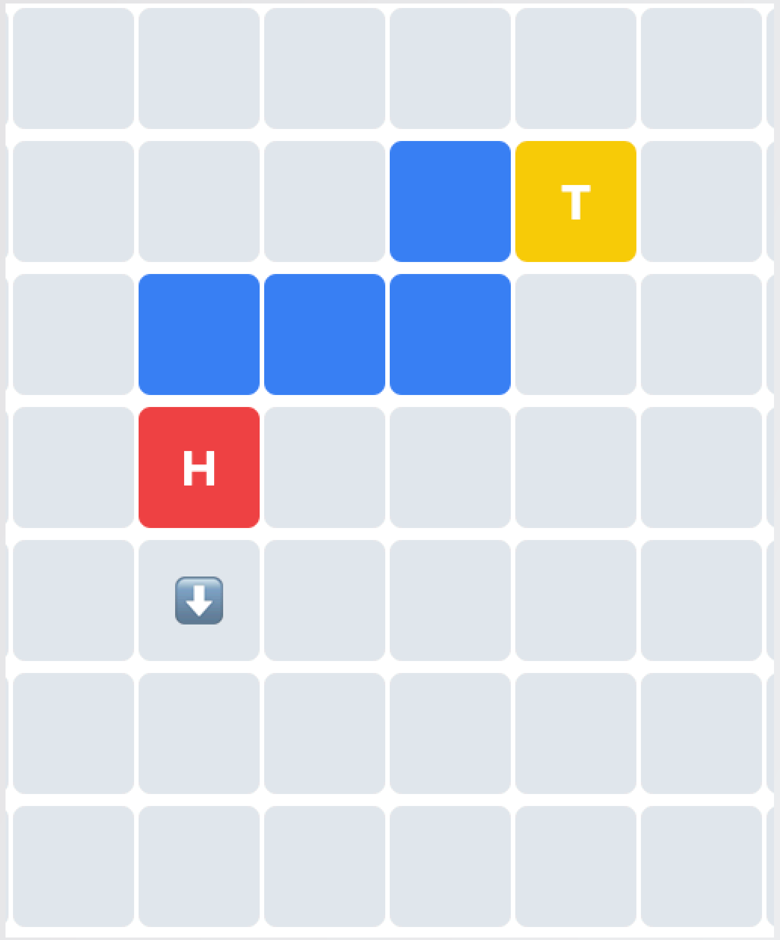
Is It Even That Deep?

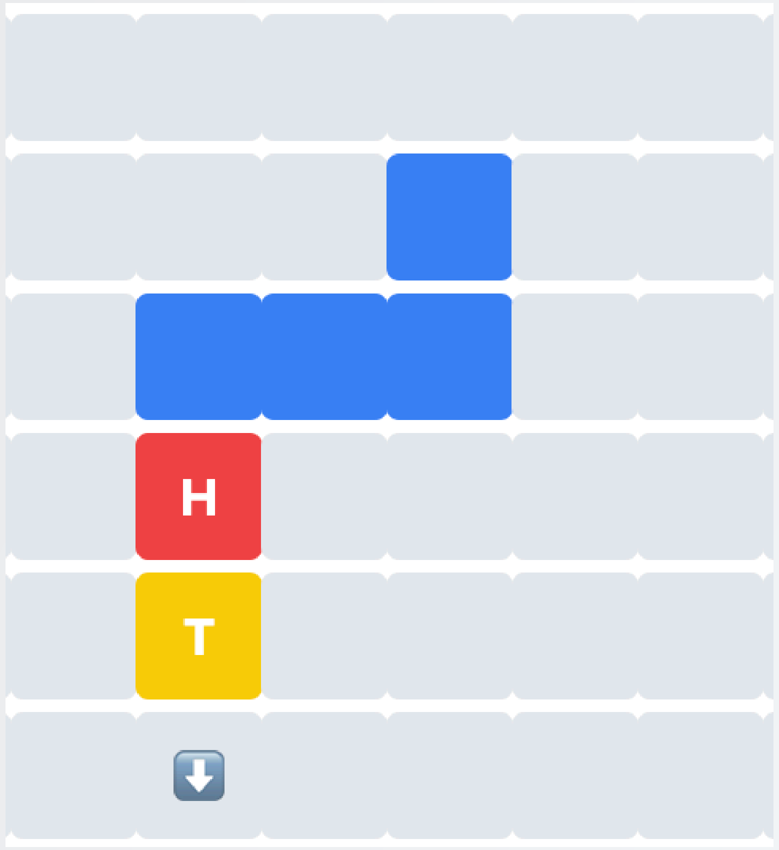
SNAKE

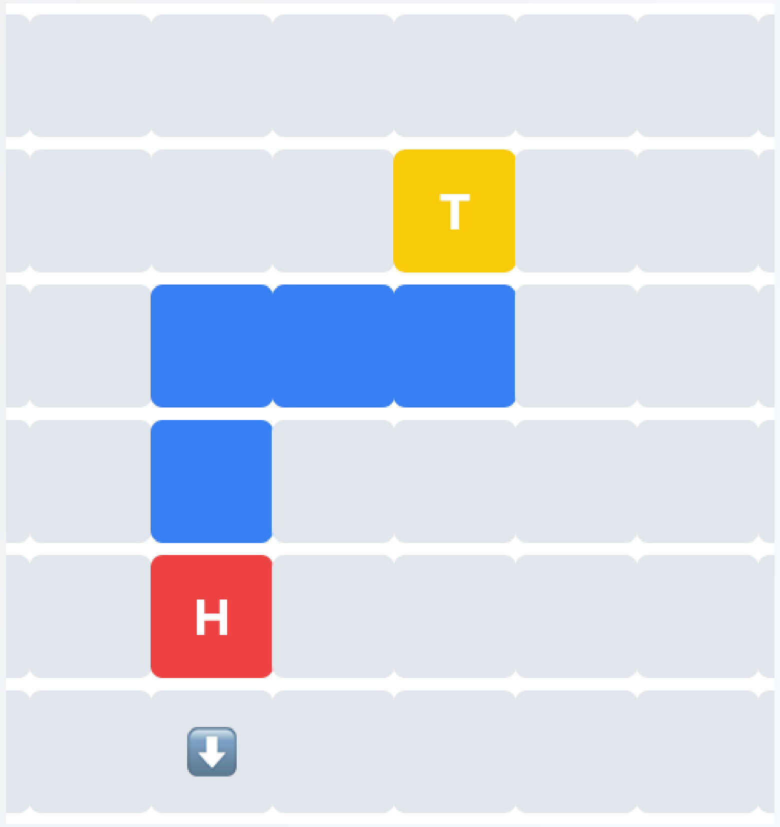




SNAKE







Moving a Snake Isn't Too Complicated...

If the segments are stored in some list-like thing (an ArrayList)...

```
function moveSnake:

    tail = segments.removeLast()
    head = segments.getFirst()
    tail.position = head.position + directionOfTravel
    segments.addFirst(tail)
```

Moving a Snake Isn't Too Complicated…

Could also represent segments as Node-like containers that store *position*, *predecessor*, and *successor*. Segments can be stored in a Tour-like sequence with a `head` and a `tail`…

In that case:

1. Save a reference to the current tail
2. Move the `tail` reference to the current `tail`'s predecessor
3. Update the `tail`'s position to be the next space to be occupied by the `head`.
4. Move the `head` reference to be the previous `tail`
5. Clean up the the old `head` and `tail` predecessor/successor references.

Don't Make a Valley Out of a Snakepit!

Coordinating Coordinates

Default PennDraw Coordinates

`(0, 0)` in the bottom-left, `(1, 1)` in the top-right.

If you want the game to be played on a grid of 15x15 cells, then…

- the height and width of each cell `0.0667`
- the half-height and half-width of each cell `0.0333`
- the center of the bottom-left cell is `(0.0333, 0.0333)`

These are gnarly numbers.

Set Scale!

Remember that `PennDraw.setScale()` exists!

We could do `PennDraw.setScale(0, 15)`. Then, divided into a 15x15 grid, we'd have:

- the height and width of each cell `1` 😃
- the half-height and half-width of each cell `0.5` 😃
- the center of the bottom-left cell is `(0.5, 0.5)` 😐

Set Scale!

We could do `PennDraw.setScale(-0.5, 14.5)`. Then, divided into a 15x15 grid, we'd have:

- the height and width of each cell `1` 😃
- the half-height and half-width of each cell `0.5` 😃
- the center of the bottom-left cell is `(0, 0)` 😃