Class #2

Plan for Class 2

#1. Critical Hit Applications
#2. Go Over Assignment 1
#3. Pseudocode practice
#4. Requirements, Use Cases, UML

Assignment #1

What is a difference between scripts in Unity and “regular” programming?
Control is passed to Unity Engine.

What is a Transform?
Transforms describe position/rotation/parents and childs of objects. Every gameobject has a transform(and visa verse).

What is MonoBehaviour?
MonoBehaviour is the base class every script derives from". Simply put: Your scripts can use all of the "MonoBehaviours" default functions.

Assignment #1

What are Awake(), Start(), Update(), FixedUpdate(), where are they called, and how are they different?

- Awake, called only once when you load a scene, before all other functions.
- Start, called at the start of a scene, after Awake.
- Update, called every frame. Use it for...anything
- FixedUpdate, called every "physics frame". Use it for physics related work.

Assignment #1

10 friends are sitting in a circle around a table and decide to play a new game. In it, they count up through the numbers from 1 to 100. The first person says "1", the second says "2" and so on... but with a few catches:

- Whenever the number is divisible by 7, they switch directions. So person 6 will say "6", person 7 will say "7", then person 6 again will say "8".
- Whenever the number is divisible by 11, they skip the next person.

Develop a program to determine which player says the number 100.

Assignment #1

It’s more important to structure the code then to get the right answer.

With coding, it’s very easy to get the right answer for the wrong reason!

Write once, deploy many times. New scenarios always come up.

Assignment #1

#1. Define the simple/base case.

-a program that counts to a maxValue (i.e.100)

-a program that determines which person (between person #1 and person #MaxPerson (i.e. 10) says the max value

-the maxValue and maxPerson may change overtime, there the code should accommodate different maxValues and maxPersons.

Assignment #1

#2. Define the tools you need

-a function with parameters

-some variables

-a “for” loop that counts from 1 to max value in increments of 1

-”something” that ensures that only 10 people can say a number.

Assignment #1

Version #1.

Test with maxPerson =10, maxNumber = 100

Test with maxPerson = 12, maxNumber = 100

```
void findTheNumber (int maxPerson, int maxNumber) {
	int counter;
	int person = 0; //person can only be between 1 and maxPerson (e.g. 10)
	for (counter = 1; counter<=maxNumber; counter++) {
		person = counter % maxPerson; //gives you the remainder.
		if(person == 0) person = maxPerson;
	}
}
```

Assignment #1

Add debug statements and Test with another maxPerson and maxNumber combo.

Assignment #1

Version 2: Add in the “switch directions” rule.

-need something to tell me when a num is divisible by a specific number (e.g. 7)

-need something to keep track over whether we are counting up or down.

-need something to keep track of the previous person and the current person that said a number.

Assignment #1

Version #2.

Test with magicNum = 7. Test with magicNum = 2.

```
void findTheNumber (int maxPerson, int maxNumber, int magicNum) {
        int counter;
        int person = 0;
        int direction = 1; //can be -1 or +1;
        for (counter = 1; counter<=maxNumber; counter++) {
           person = (person + direction) % maxPerson;
           if(person == 0) person = maxPerson;
            //switch directions whenever the counter is divisible by a magic number
            if(counter % magicNum ==0) direction = direction * -1;
        }
}
```

Assignment #1

Version 3: Add in the “skip” rule.

-need something to tell me when a num is divisible by a specific number (e.g. 11)

-need something to keep track over whether we are counting up or down.

-need something to keep track of when we skip a person.

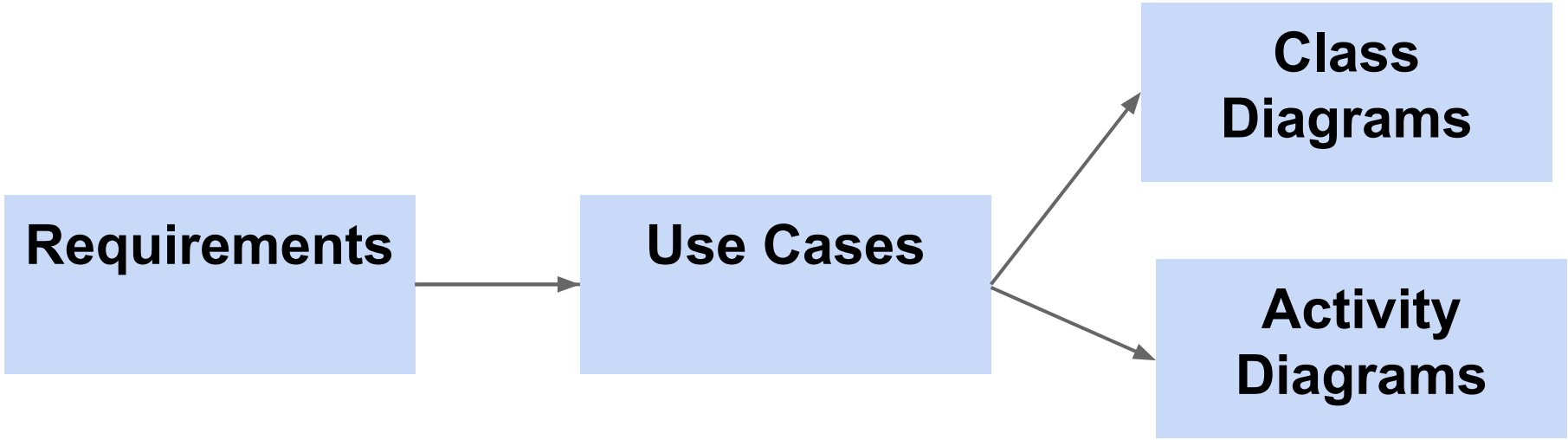
Assignment #1

Version #3.

Test with magicNum1 = 11. Test with magicNum1 = 4.

```
int skip = 0;
person = (person + direction + skip) % maxPerson;

//skip a number whenver the counter is divisible by another magic number
	if (counter % magicNum1 == 0) {
		skip = direction;
		}
	else skip = 0;
```


Requirements
Use Cases
Class Diagrams
Activity Diagrams

Week Breakdown

Week 1: Requirements, Core Use Cases, UML

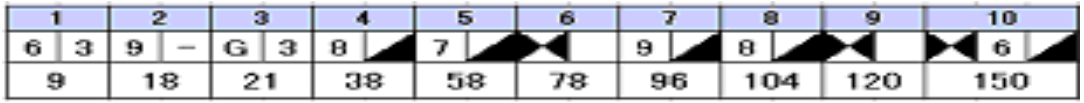
Week 2: Additional Use Cases, Class Design using UML.

Week 3: Coding for Core Use Cases.

Week 4: Code for Additional Use Cases.

Bowling Game

Requirements



1	2	3	4	5	6	7	8	9	10
6 3	9 –	G 3	8 /	7 / X		9 /	8 / X		X 6 /
9	18	21	38	58	78	96	104	120	150

Bowling Score Requirements

- Show the track of a bowling score once the game has started.
- Update the score every time the player throws the bowl.
- Players can change the scores during the game.
- Show an animation when someone made the strike.

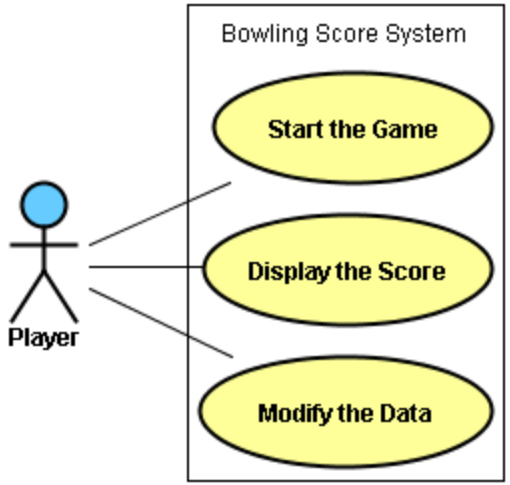
Use Cases And Actors

Use Cases describe how a user uses a system to accomplish a particular goal.

A Use Case is a list of steps, typically defining interactions an “actor” and a system, to achieve a goal.

The actor can be a human, an external system, or time.

Use Cases And Actors



UML Basics

UML is a general purpose visual modeling language to visualize, specify, construct and document software systems.

UML is powerful enough to represent all the concepts exists in object oriented analysis and design.

Object Oriented Design - Recap

UML diagrams are representations of object oriented concepts only. Following are some fundamental concepts of object oriented world:

- **Objects:** Objects represent an entity and the basic building block.

- **Class:** Class is the blue print of an object.

- **Abstraction:** Abstraction represents the behavior of an real world entity.

Object Oriented Design - Recap

- **Encapsulation:** Encapsulation is the mechanism of binding the data together and hiding them from outside world.

- **Inheritance:** Inheritance is the mechanism of making new classes from existing one.

- **Polymorphism:** It defines the mechanism to exists in different forms.

UML Basics

UML plays an important role in defining different perspectives of a system. These perspectives are:

- Design
- Implementation
- Process
- Deployment

UML Basics

- **Design** of a system consists of classes, interfaces and collaboration. **UML provides class diagram, object diagram to support this.**

- **Implementation** defines the components assembled together to make a complete physical system. UML component diagram is used to support implementation perspective.

UML Basics

Process defines the flow/activity of the system. So the same elements as used in *Design* are also used to support this perspective.

Deployment represents the physical nodes of the system that forms the hardware. UML deployment diagram is used to support this perspective.

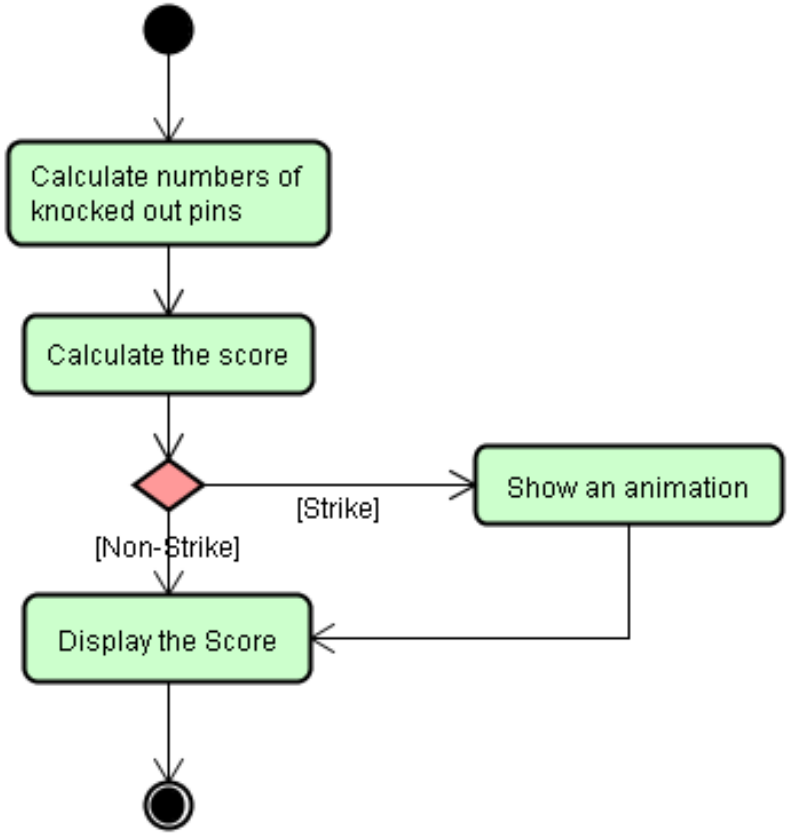
Activity Diagrams

An Activity diagram is basically a flow-chart to represent the flow from one activity to another activity.

The activity can be described as an operation of the system.

It captures the dynamic behaviour of the system

Activity Diagrams



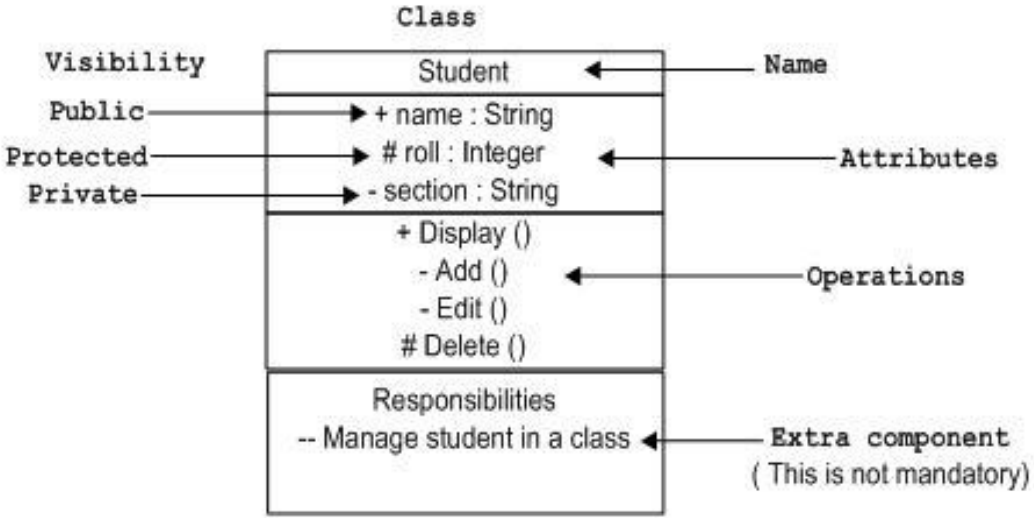
UML Basic Notations

UML Class Diagrams

UML *class* diagram is divided into four parts.

- The top section is used to name the class.
- The second one is used to show the attributes of the class.
- The third section is used to describe the operations performed by the class.
- The fourth section is optional to show any additional components.

Classes


Class
Visibility
Student
Name
Public
+ name : String
Protected
roll : Integer
Attributes
Private
- section : String
+ Display ()
- Add ()
Operations
- Edit ()
Delete ()
Responsibilities
-- Manage student in a class
Extra component
(This is not mandatory)

Objects

What is an object?

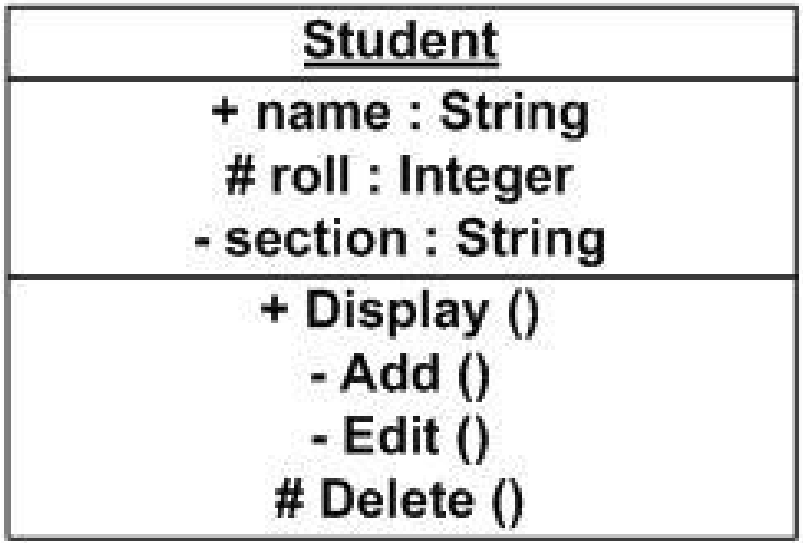
Objects

As object is the actual implementation of a class which is known as the instance of a class. So it has the same usage as the class.

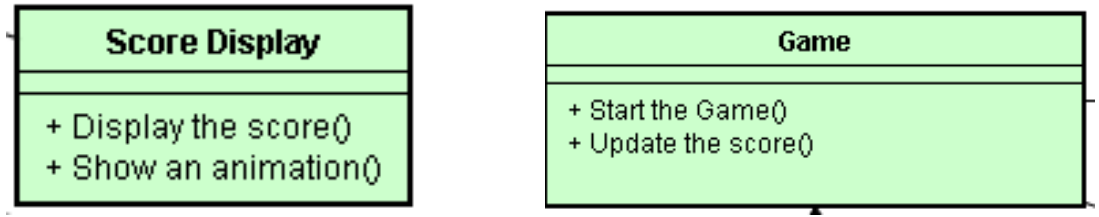
An *object* is represented in the same way as the class.

The only difference is the *name* which is underlined as shown below.

Objects


Student
+ name : String
# roll : Integer
- section : String
+ Display ()
- Add ()
- Edit ()
Delete ()

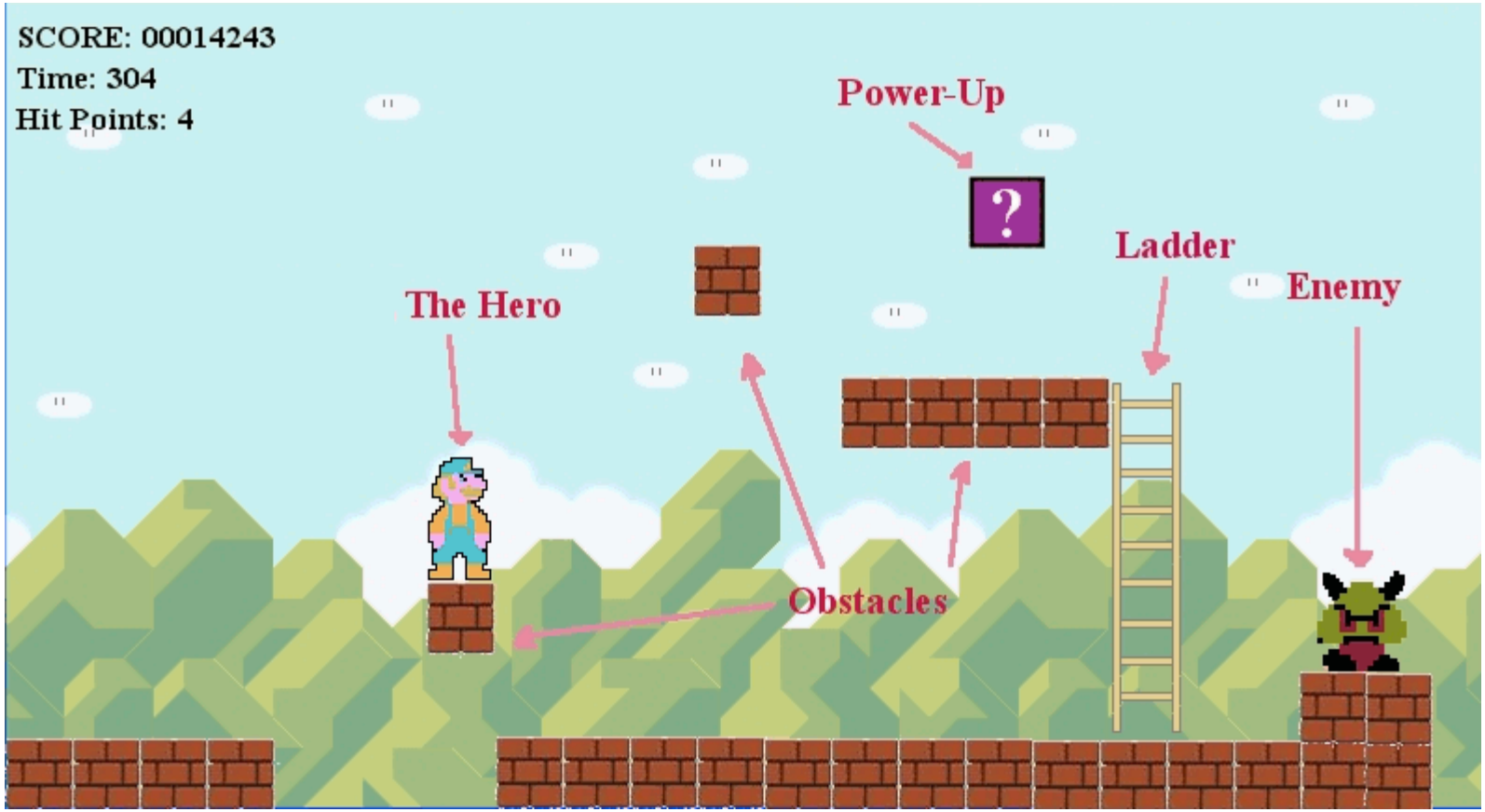
Bowling Game Classes



2D Platformer

2D Platformer - Sample Project

GOAL: Create a single-player, 2D, side-scrolling platformer with obstacles, enemies, levels, and items, inspired by Mario.


SCORE: 00014243
Time: 304
Hit Points: 4
Power-Up
?
Ladder
Enemy
The Hero
Obstacles

Requirements

- The game will have a scrollable screen that moves from left to right along with some up and down scrolling.
- The player controls a character, who can walk/run, jump over obstacles, attack enemies, and collect items.
- The goal is to get to the end of each level.
- The score will be tallied as the game is played. Score increases with killed enemies and collected items.

Requirements

- The character has "hit points" which are lost when he is attacked by an enemy or when he is hit by falling objects.
- When an enemy is killed (hit points reduced to zero), it stays killed.
- Physics will be applied to the character when jumping off a platform.
- The level is designed with the objects placed on a grid, the characters and objects can move independent of the grid.

Requirements

What other requirements should we add?

Actors

- **Player:** Can Save/Load game or start a new game. Controls the character via the keyboard. Can also pause the game.

- **Character:** The hero of the game. Can move according to controls given by the player. He/She can impeded by obstacles, collect power ups, and attacks enemies. He/She also loses hit points if attacked by an enemy or hit by a falling obstacle. He/She is killed when his hit points reaches zero.

Actors

- **Obstacle:** Depending on the type of obstacle, it can either impede the Character's movements or decrease his hit points.

- **Enemies:** Can attack the Character, determining his hit points, or can be attacked by the Character and have their hit points reduced, getting killed when their hit points reach zero.

- **Power Ups:** Increase the Character's hit points and the game score when collected. Also might give him a better weapon, or increase his size.

Use Cases

Use Cases describe how a user uses a system to accomplish a particular goal.

User Opens game

- Can start new
- Can play saved game

Character Collects an item

- Depending on the item…
- Score increases

Character Attacked by Enemy/Obstacle

- Reduce hit points/stop movement

Use Cases

What are Other Use Cases?

Use Cases

- Character Jumps off Platform
  - Appropriate physics dictate movement

- Character Attacks Enemy
  - Reduce Enemies hit points
  - Score increases

- Enemy's Hit Points Reduced to 0
  - Enemy Dies
  - Enemy remains dead

Use Cases

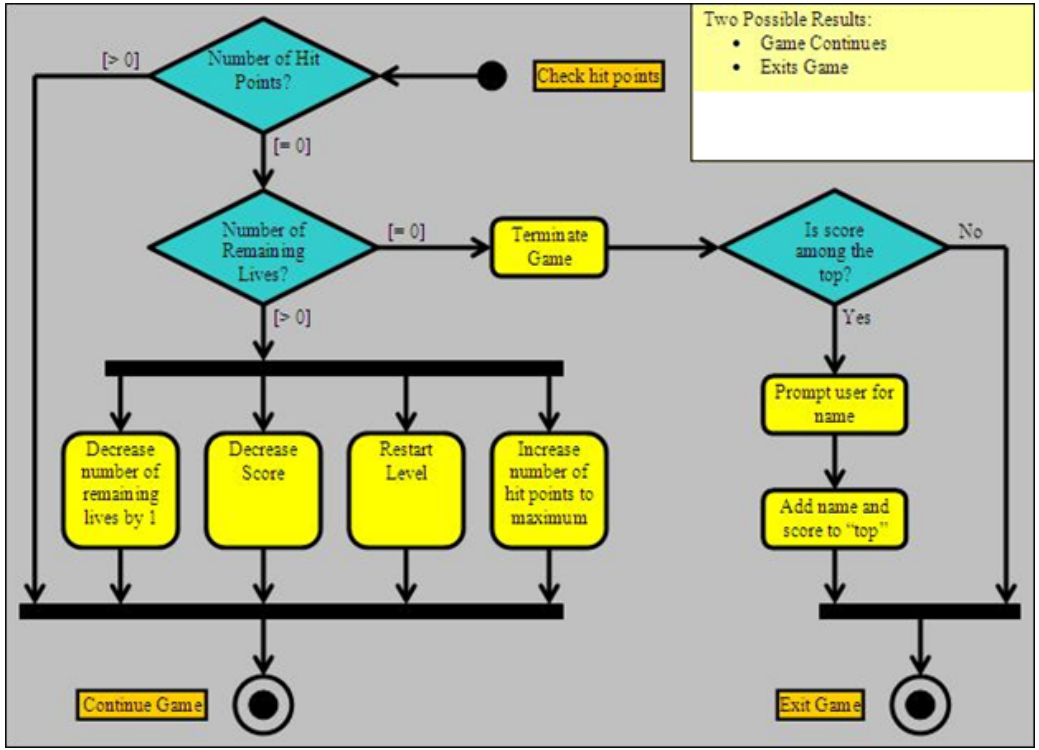
- Character's Hit Points Reduced to 0
  - If Character has remaining lives:
    - Character Dies
    - Score decreases
    - Level restart
  - If Character doesn't have remaining lives:
    - Game over
    - User can enter name if score among the top

Use Cases

- User reaches the end of screen
  - Next level loads

- User Saves Game
  - Game saved

Activity Diagram



Activity Diagram

- Select a Use Case and draw an activity diagram!