FALMOUTH UNIVERSITY

4: UML – Unified Modeling Language

GAM140: Individual Creative Computing Project

Assignment Roadmap

- **Assignment 1**
  - Week 6 – First prototype of game and controller
  - Week 9 – Peer review of game and controller
- **Assignment 2**
  - Week 3 – Project Proposal
  - Week 8 – Draft Poster presentation
  - Week 10 – Report Peer Review

- **Next up: WEEK 6 – First Prototype of game and controller**

Learning outcomes

- **Understand** rationale behind UML
- **Understand** a subset of UML Diagrams useful for game development
- **Develop** some UML Diagrams

Introduction

- In COMP110 you were introduced to flow charts and pseudocode
- These were useful for designing the high level flow of an application, and detail how an algorithm could be implemented
- UML is an attempt to create a formal design language for designing software

What is UML?

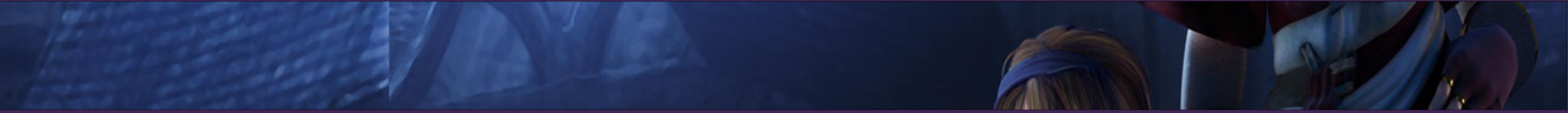
- UML is a visual notation system which can be used to design software
- It was first devised in 1996 by Booch, Jacobson and Rumbaugh
- The goal was to unify/standardise all the various modelling languages and diagrams used in Software Development
- In 2005, ISO published UML as an international standard
- UML 2.0 is the most current version, there are currently 14 different diagram types

Why UML?

- UML offers us a standardised way of designing software
- It allows us to think through our systems before committing them to code
- It offers a shared language between programmer and other disciplines including clients

Diagram Types

- UML2.0 is split into two diagram families
- **Behaviour Diagrams**
  - – Describes what happens in a system, this includes interactions between users and the system
  - – Or the current system and other external systems
- **Structure Diagrams**
  - – Describes what is contained in the system
  - – Typically used to model the system



Diagramming Tools

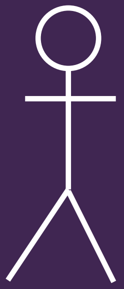
- Gliffy - https://go.gliffy.com/go/auth/login
- draw.io - https://www.draw.io/
- Microsoft Visio - https://products.office.com/en-gb/visio/flowchart-software

BEHAVIOURAL DIAGRAMS

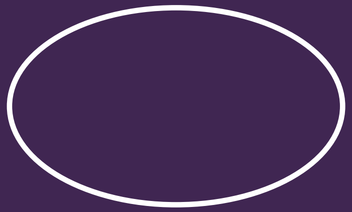
Use Case Diagram

- Use Case diagrams typically details the user's interaction with the system
- In essence it details the **Use Case** of the system and the **Actors** which interact with the system
  - **NB.** These **Actors** could be other systems!
- Created using terms that a layman could understand
- Can be used to capture and communicate User Requirements
- This is often the first diagram created for a system

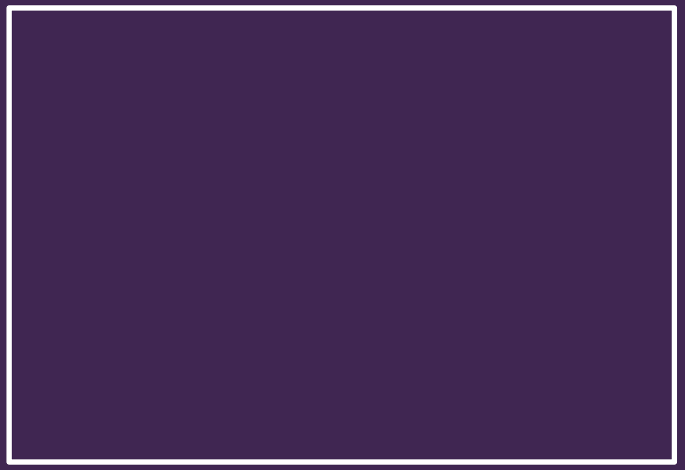
Use Case Diagram



Actor

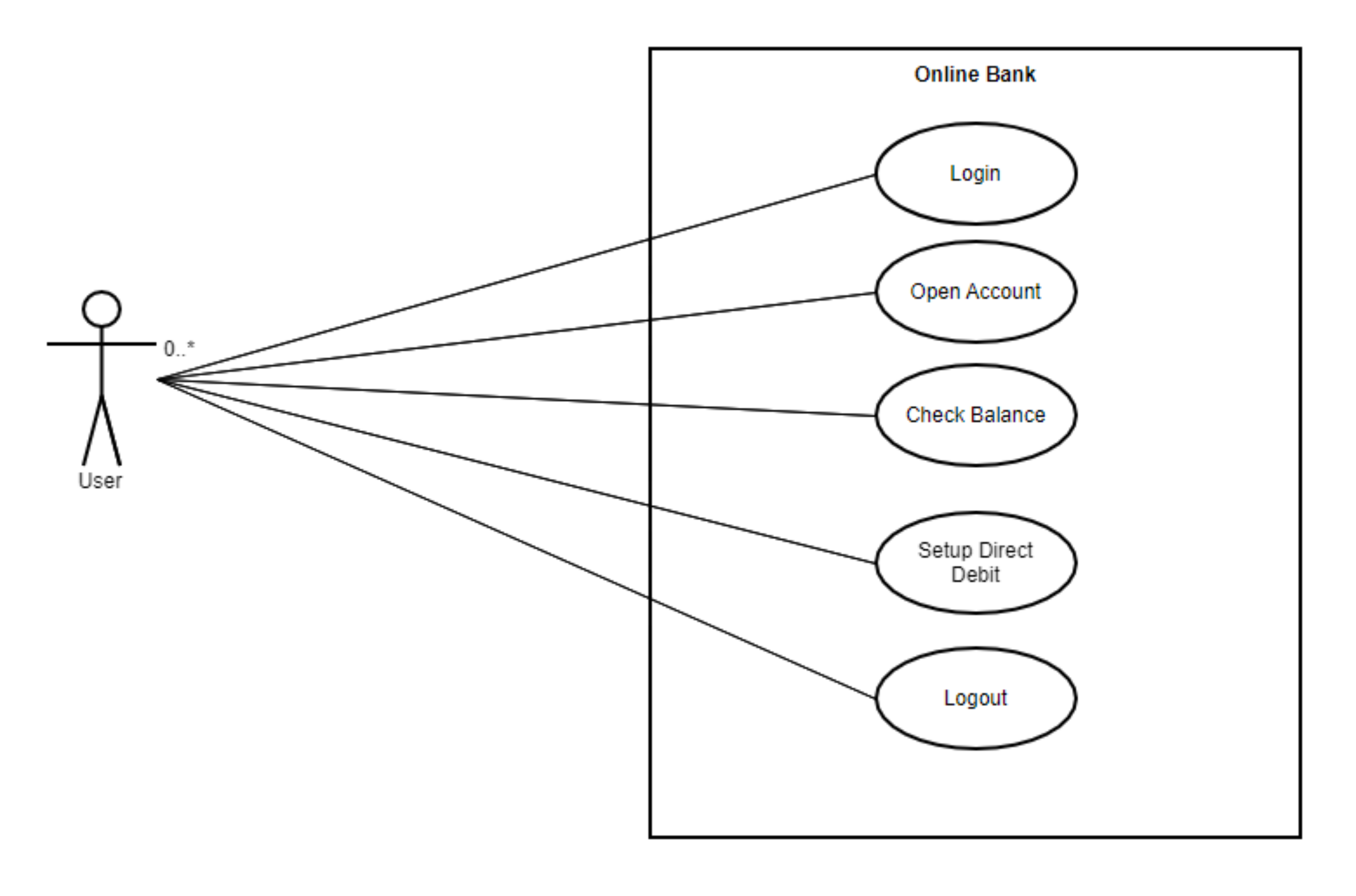


Use Case



System Boundary

Use Case Diagram



Use Case Diagram – Class Exercise

- What are the key **Use Cases** for **Discord**?
- **Lets diagram it together**

Activity Diagram

- Activity Diagrams describe behaviour composed of a collection of tasks
- This is used to model the flow of work and/or data in a system
- This type of diagram supports choice, iteration and concurrency
- You can think of this diagram as a structured Flow Chart

Activity Diagram

● Start Node

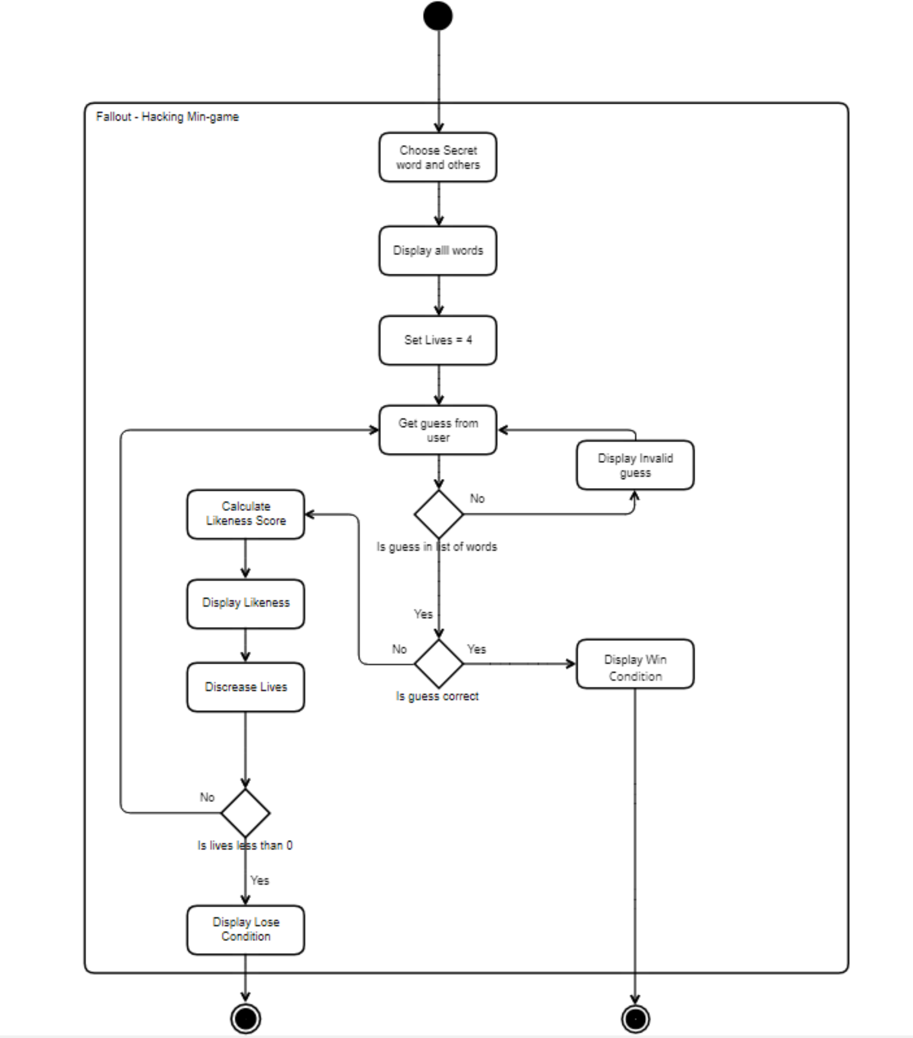
◉ End Node

◇ Decision

▭ Actions

▬ Split or join of concurrent activities

Activity Diagram



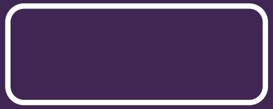
State Diagram

- State Diagrams are used to model the possible states of your applications
- This allows you to not only to model the states but the flow of events and transitions between states
- Is useful for modelling the following in games:
  - AI Finite State Machines
  - Game States
  - Animation Systems

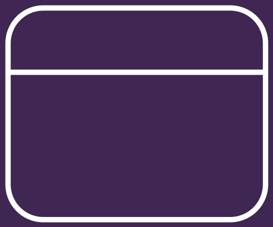
State Diagram

● Initial State

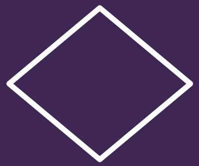
◉ Final State



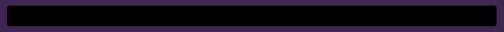
Simple State



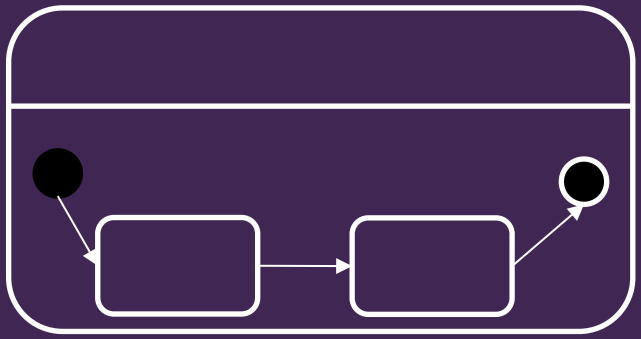
Simple State with compartments



Choice

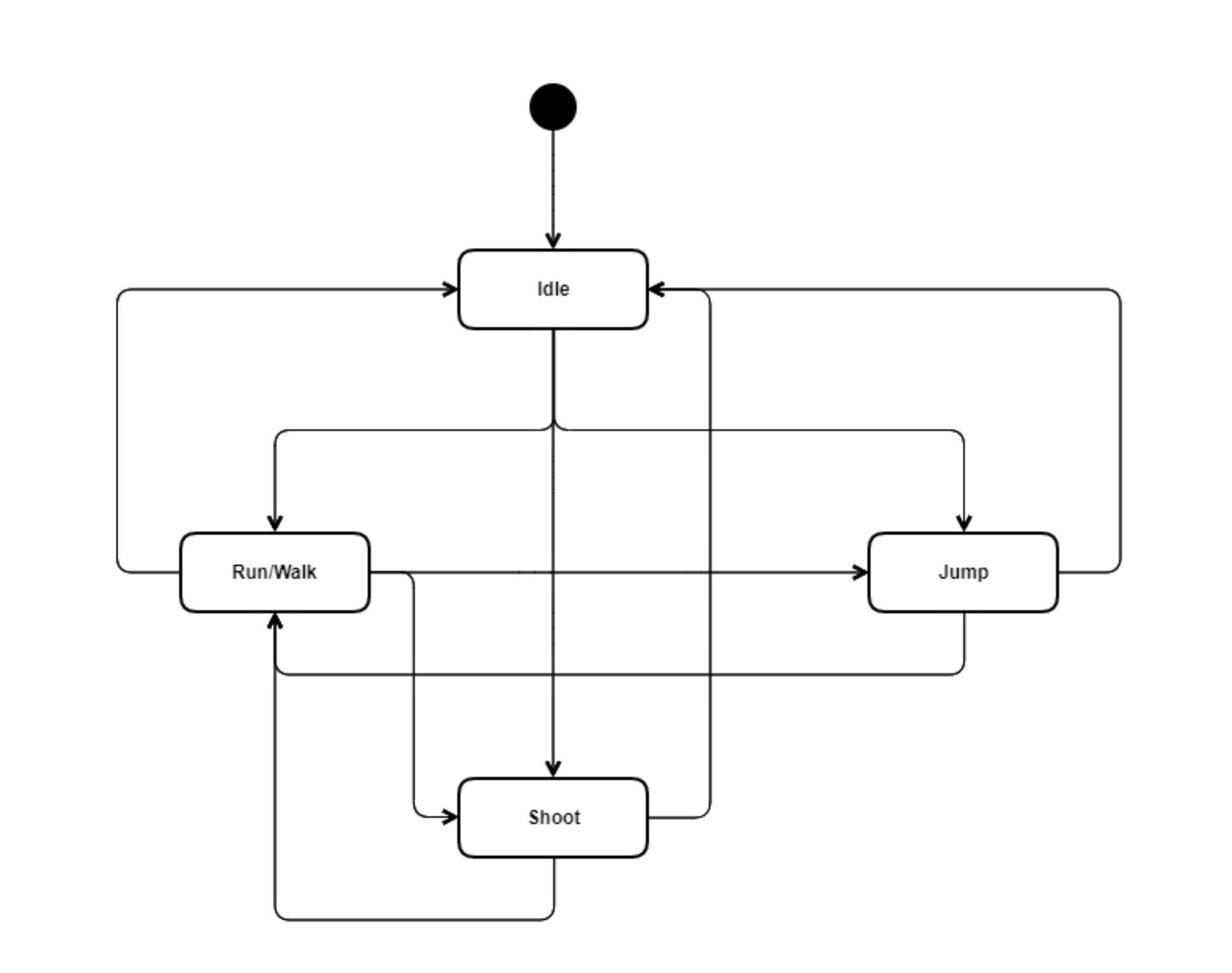


Fork or Join



Composite State

State Diagram



State Diagram – Class Exercise

- Watch the following video

https://www.youtube.com/watch?v=gMk7WIGi-W0

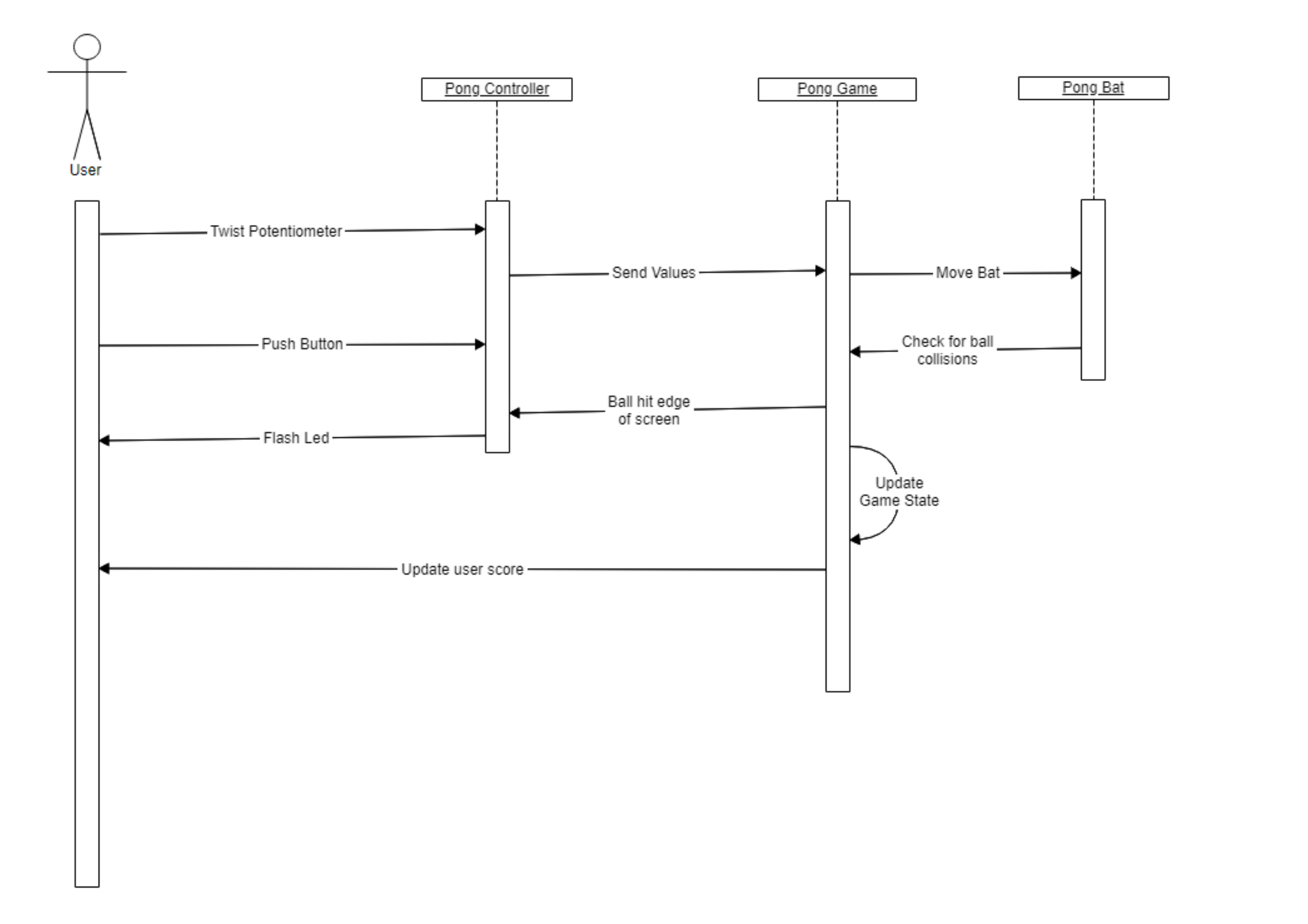
State Diagram – Class Exercise

- **Identify the States of the Cathedral Grave Warden**

Sequence Diagram

- This can be used to model the flow of logic in a system
- This is useful to see how the user interacts with the system
- How the data flows between different parts of the system
- These diagrams are often time focused with the vertical axis used to represent time

Sequence Diagram

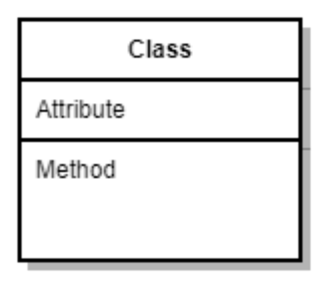


STRUCTURAL DIAGRAMS

Class Diagram

- This attempts to model object-orientated systems
- Is the one diagram which can be directly translated into code
- It has entities which represent:
  - Classes with functions and variables
  - Interfaces
  - Enumerations
- It can also be used to model relationships between classes
  - Dependency
  - Association
  - Aggregation
  - Composition
  - Inheritance
  - Realization/Implementation

Class Diagram

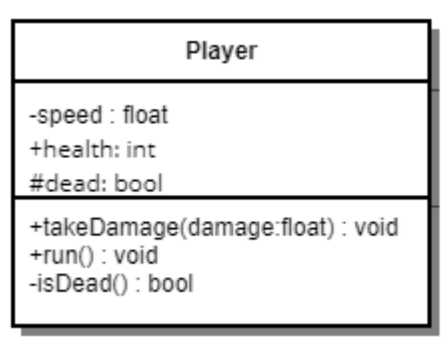


Class

Attribute

Method

Class

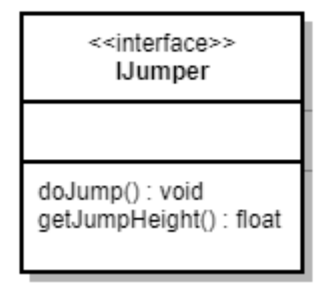


Player

-speed : float
+health: int
#dead: bool

+takeDamage(damage:float) : void
+run() : void
-isDead() : bool

Class (example)
- Private
+ public
# protected

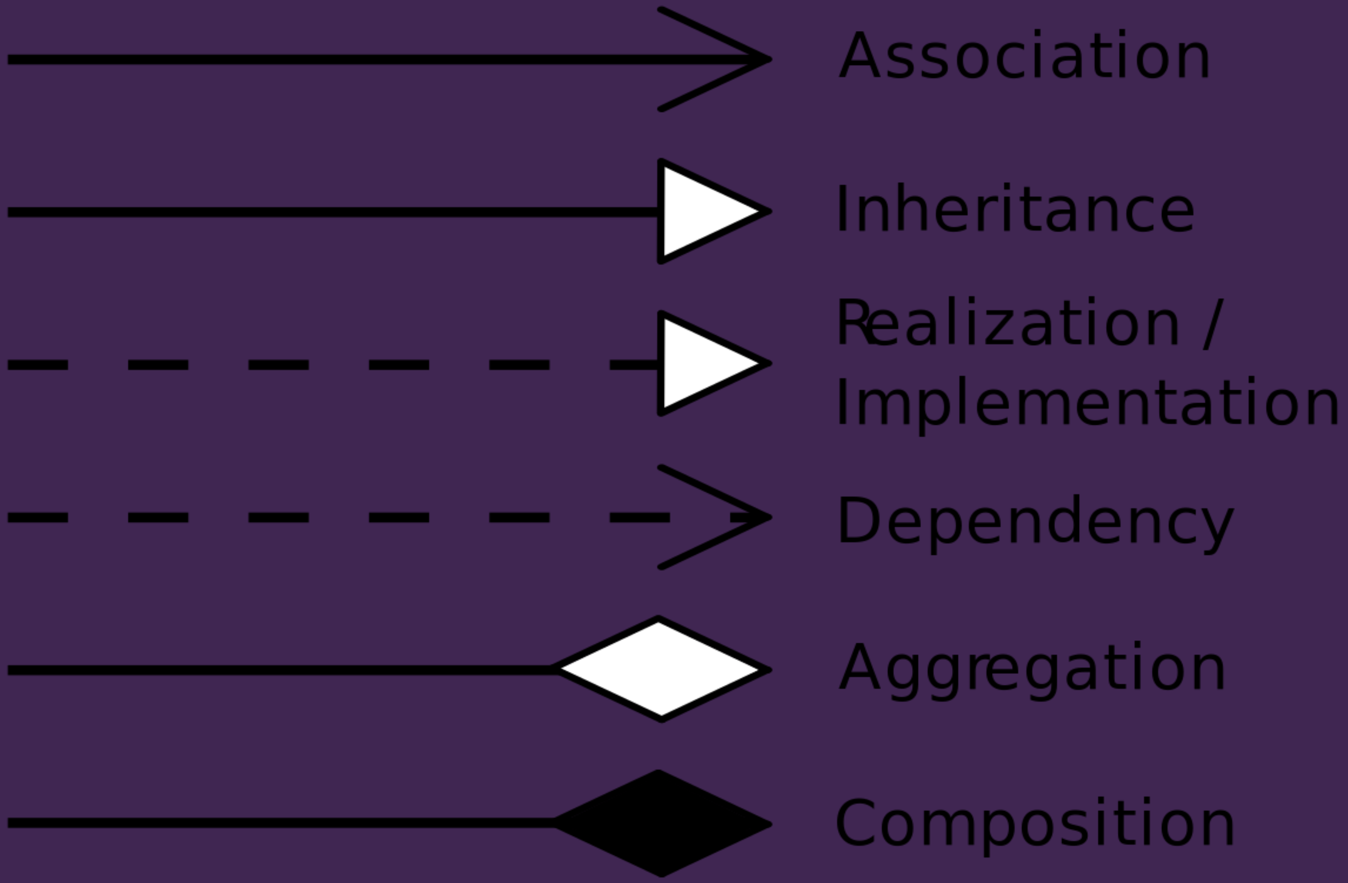


<<interface>>
IJumper

doJump() : void
getJumpHeight() : float

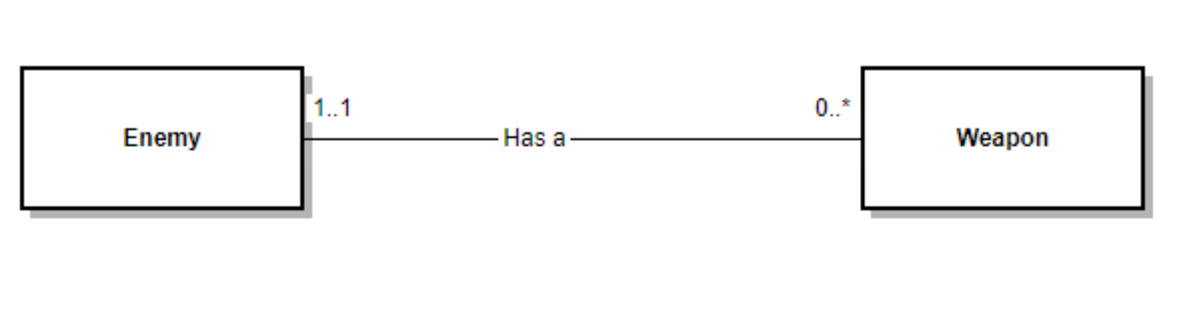
Interface

Class Diagram



https://en.wikipedia.org/wiki/Class_diagram

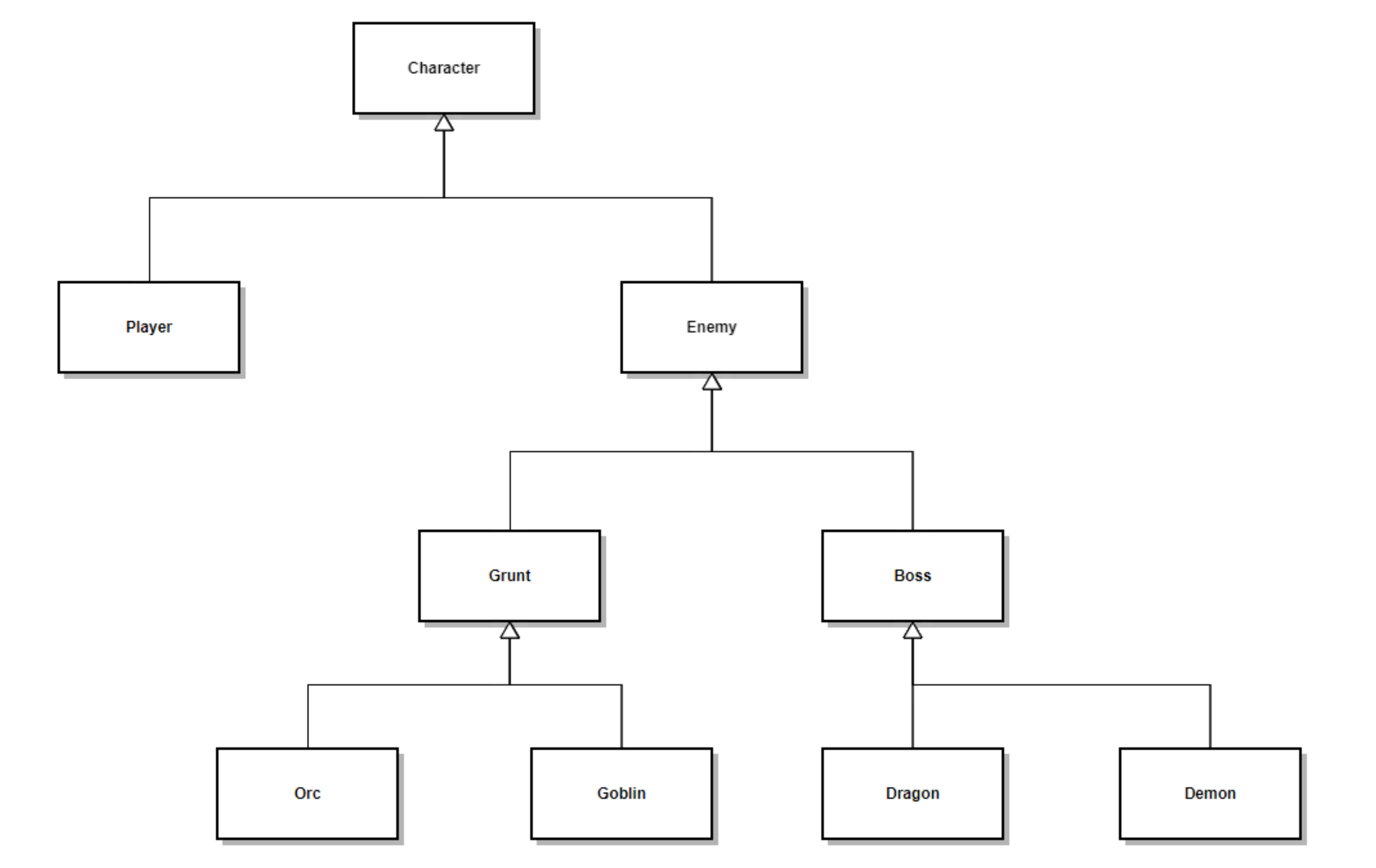
Class Diagram



Association:

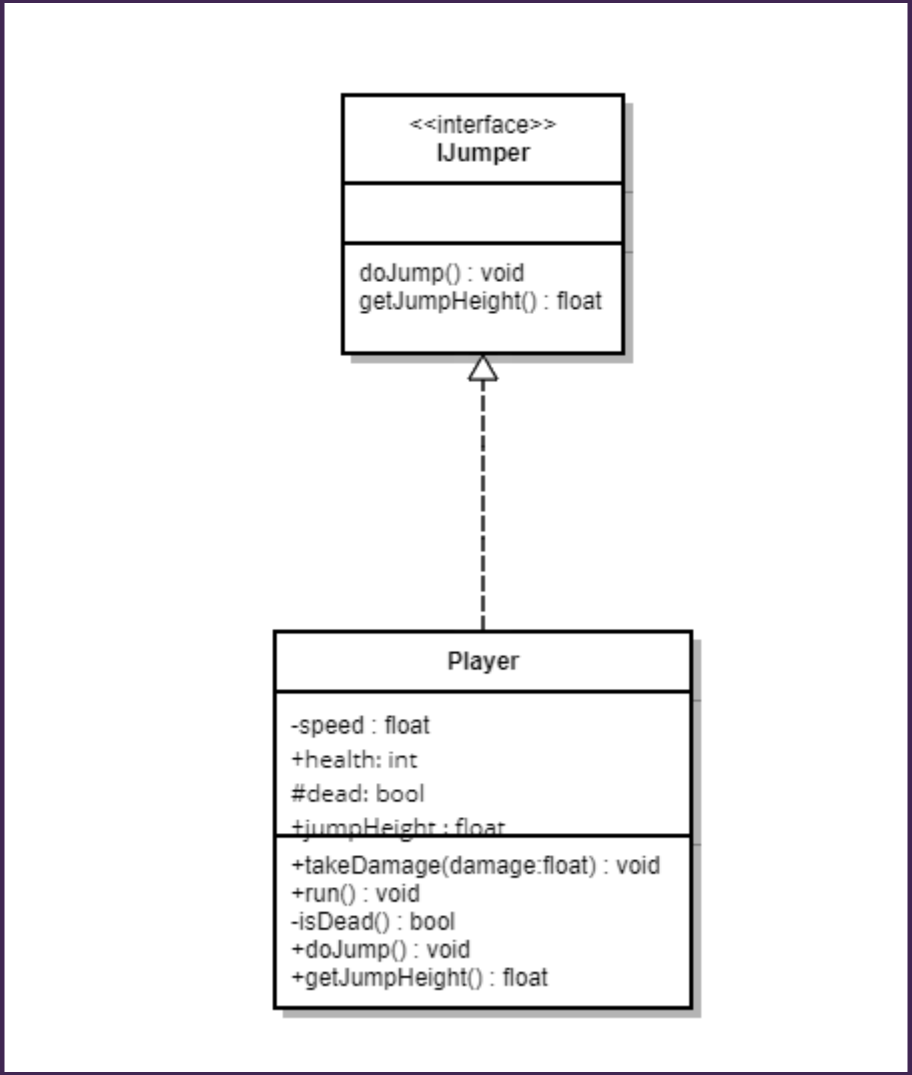
- Enemy has **0** or **many weapons**
- A **Weapon** has only 1 Enemy

Class Diagram



Inheritance

Class Diagram



<<interface>>
IJumper

doJump() : void
getJumpHeight() : float

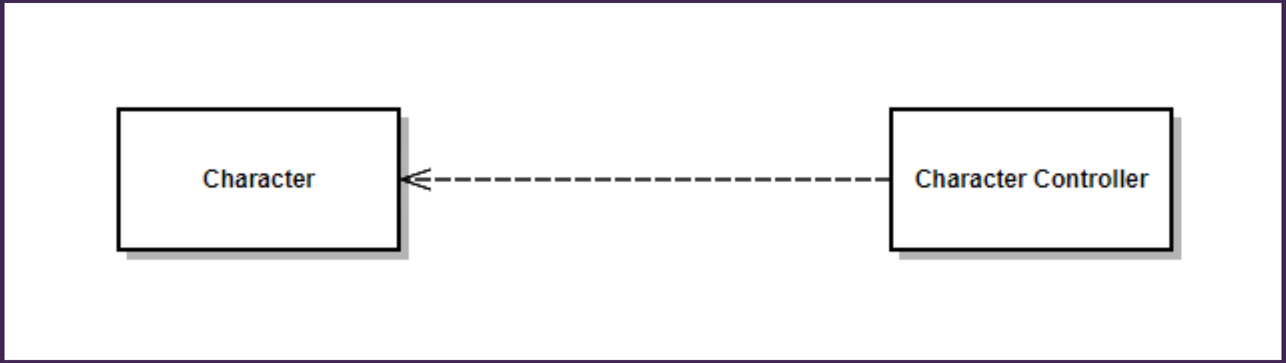
Player

-speed : float
+health: int
#dead: bool
+jumpHeight : float

+takeDamage(damage:float) : void
+run() : void
-isDead() : bool
+doJump() : void
+getJumpHeight() : float

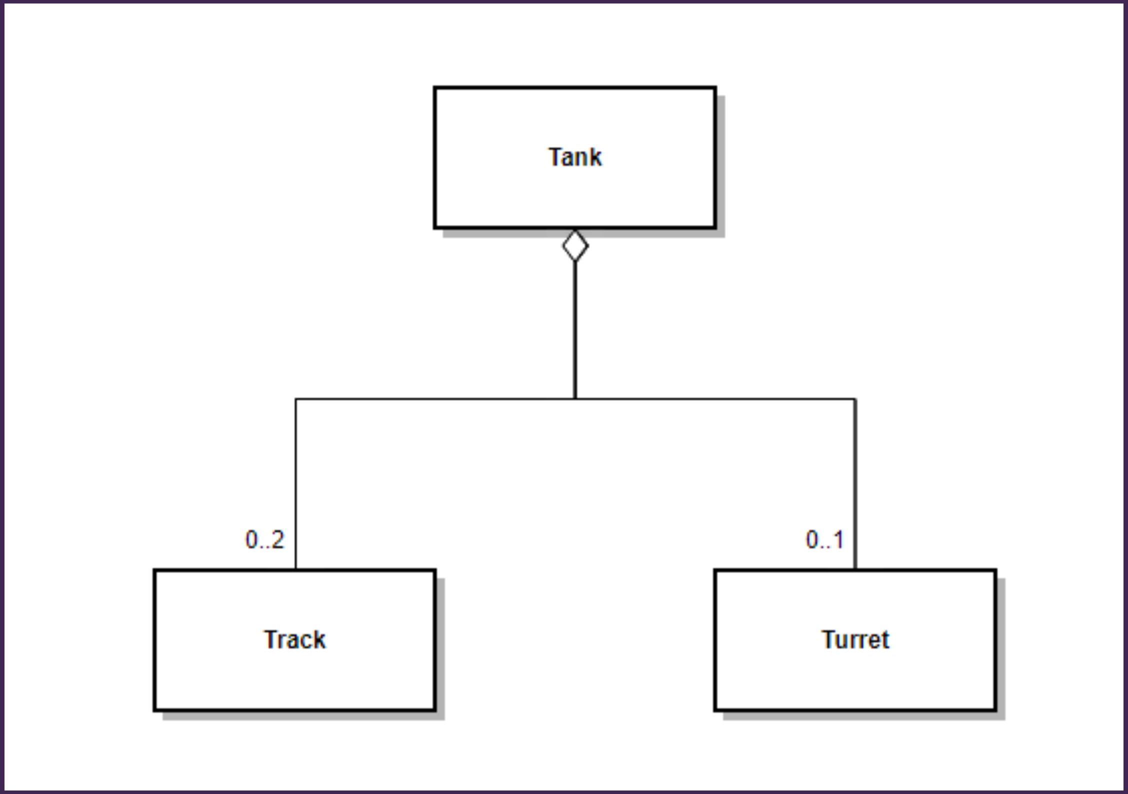
Implements

Class Diagram



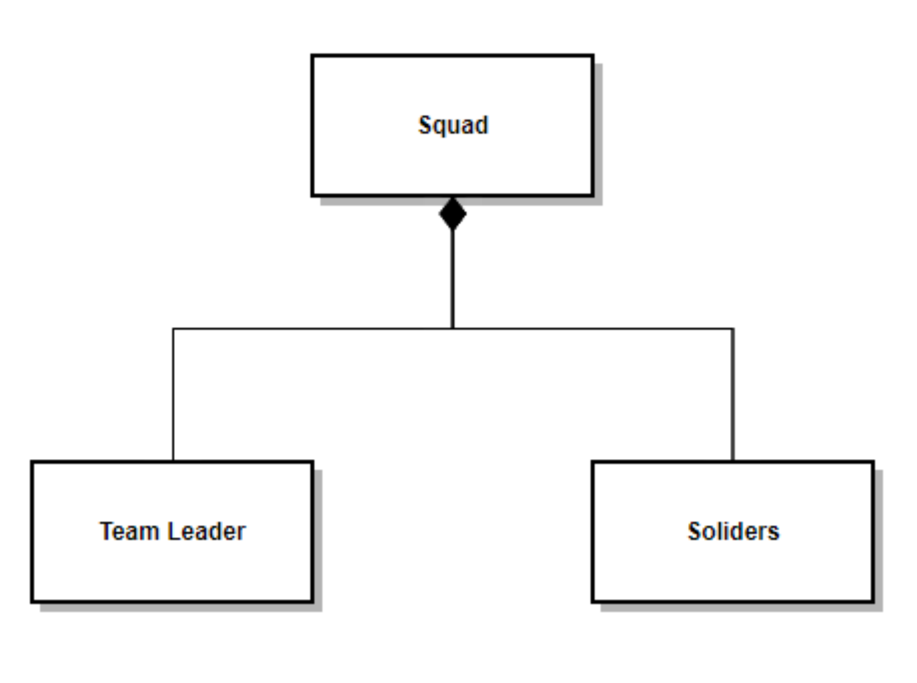
Dependency

Class Diagram



Aggregation

Class Diagram



Composition

UML Tips

- While UML is a standard, like Agile it is sometimes helpful to modify for your use case
- You can make multiple diagrams at different levels
  - A high level class diagram to show relationships
  - Lower level which shows implementation
- You don't need to use each diagram type in your projects, you will find some more useful than others