Object Oriented Software Design ISYS1083/1084

Assignment 2: Human v Alien

Lecturer Caspar Ryan
Email caspar.ryan@rmit.edu.au

Tutor Halil Ali
Email halil.ali@rmit.edu.au

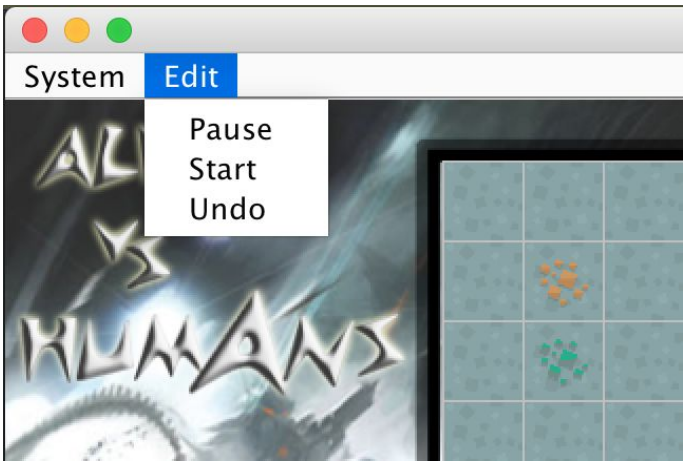
Tutor Keith Foster
Email keith.foster@rmit.edu.au

Group members:

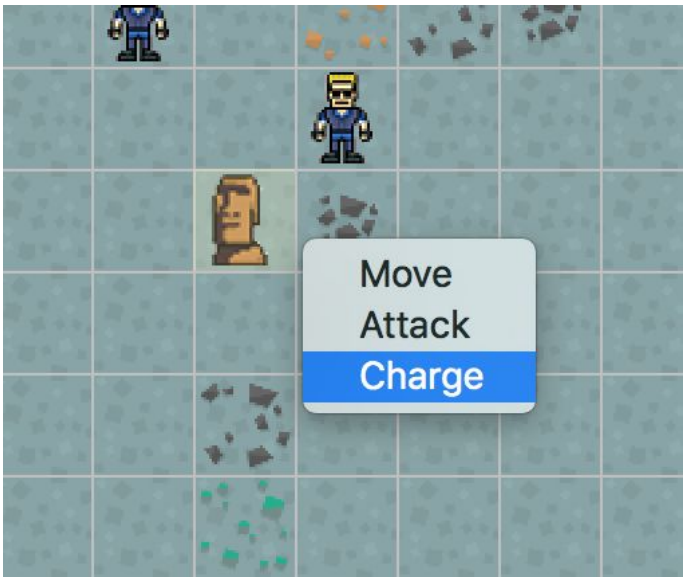
Chang Yi Wu s3518495@student.rmit.edu.au
Sai Zhang s3297071@student.rmit.edu.au
Karleen Heong s3523549@student.rmit.edu.au
Qiuwen Yao s3492704@student.rmit.edu.au

B. Additional Functional Requirements

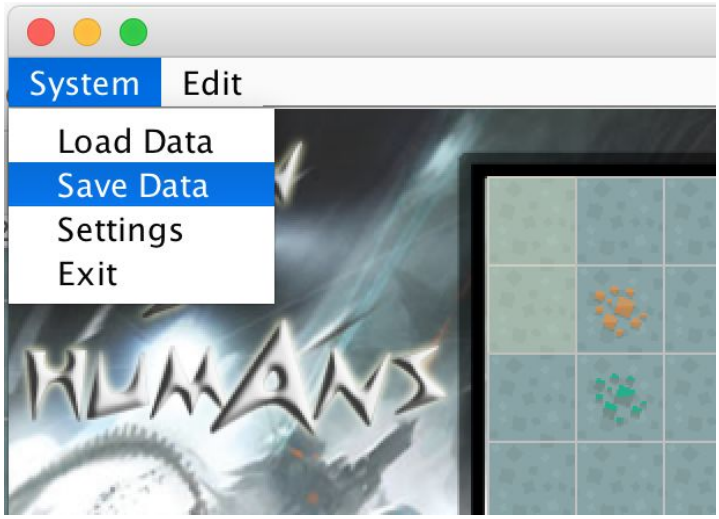
- Incorporate an undo moves option. Each player can undo from 1 to 3 moves in one go but can exercise this option only once per game. Undoing affects both players regardless of who initiated it. For example if Player 1 undoes the last two moves then both players last two moves are cancelled and Player 1 now plays their turn.



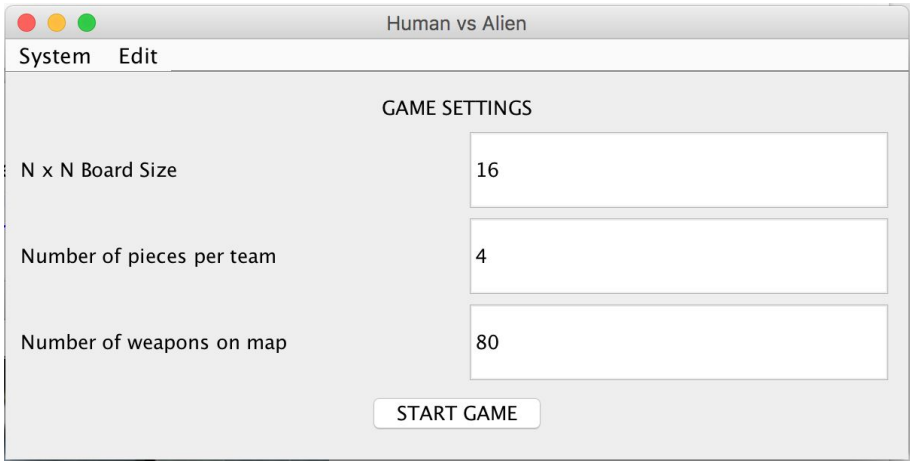
- Add an additional combat capability to each piece (players select the mode e.g. fighting stance versus defensive stance) before each move.



- An option to save the game state midway and restart later. (Before loading game, the size of game board must be identical to when it saved. For example, if game saved as a 10x10 size of board. Before loading game, user must adjust System->Settings-> board size to 10)



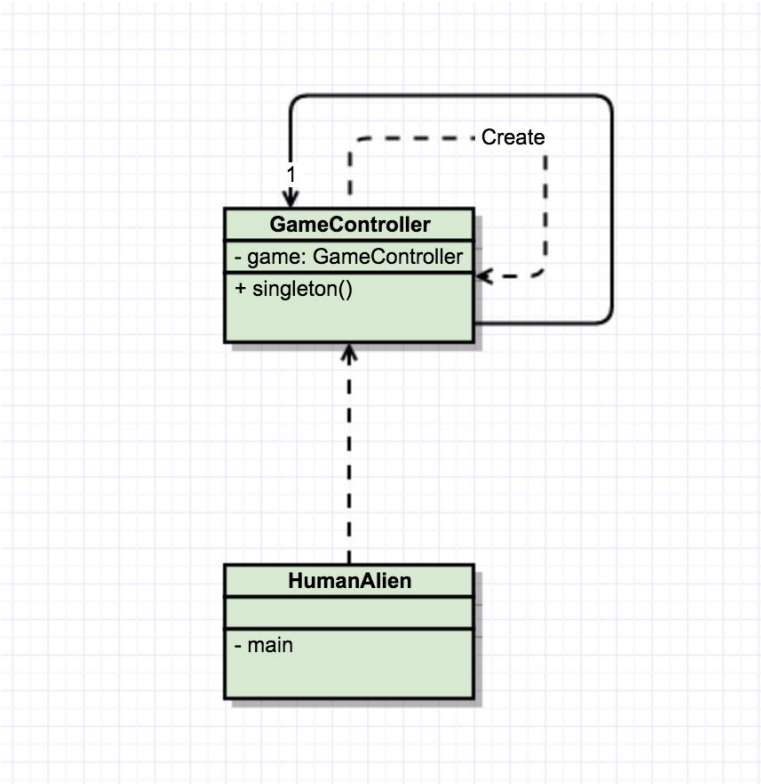
- An option to create different sized boards and varying number of pieces.



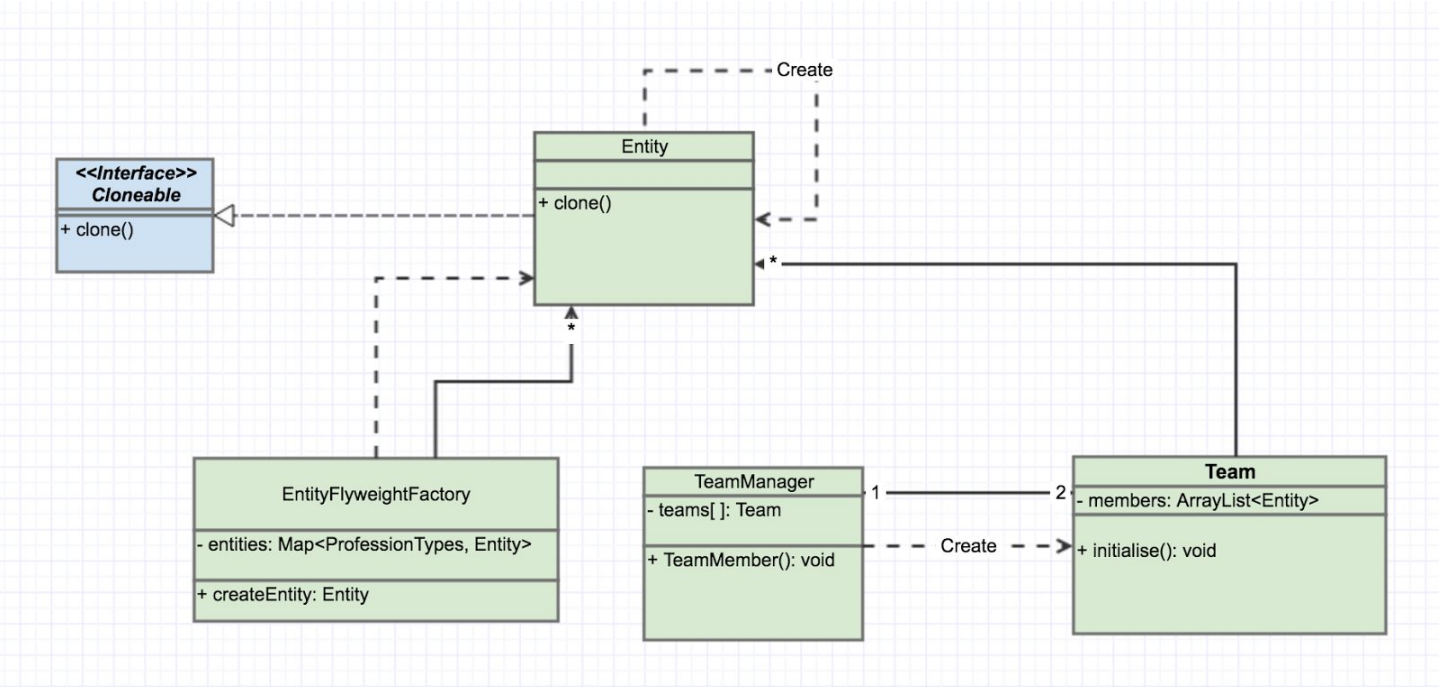
Non Functional Requirements (Based on design/code)

Creational Patterns

- **Singleton:**
- ensure a class only has one instance, and provide a global point of access to it, only have one instance, easily accessible.
- This GameController class is the controller to handle UI events, process the axial parameters and pass them to MainGame Singleton pattern is implemented here, because only this controller seated in between view and model. Singleton to ensure Main view only can create once. It responsible for the initialization of GameBoard.



- **Prototype:**
- specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype. The Entity class may have the unknown function or objects.
- Entity is a basic profession class. All profession classes in the game extends this class.Prototype. This class can be initialized to be a prototype. A basic entity can be transform to a advanced profession object through decorate pattern. When a basic entity picked up a weapon on the map

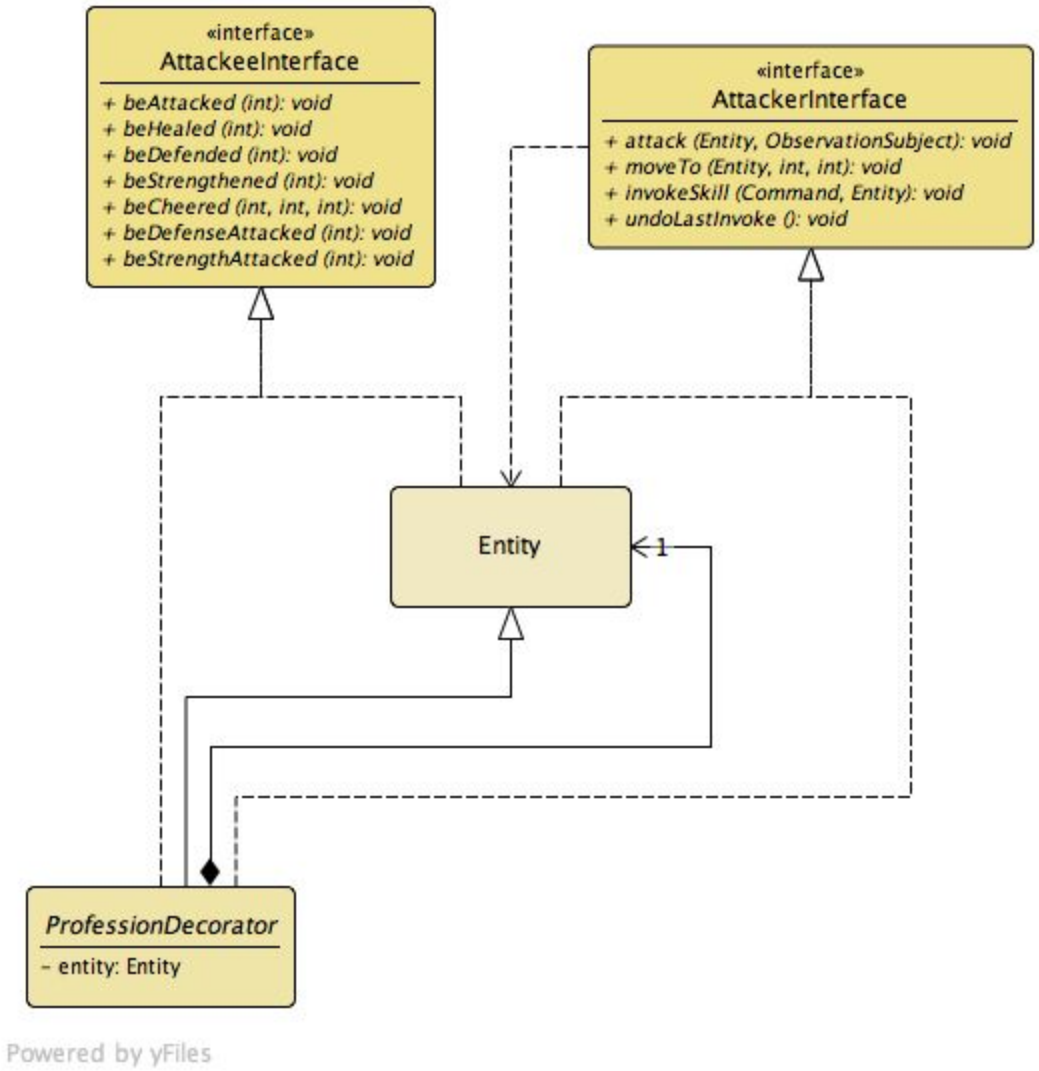


- **Factory:**
- Please refer to flyweight pattern. In our assignment, we implemented a flyweight factory to responsible for creation of entity. For GameBoard class, it does not know the type of class that an entity will be, so we make a factory for it to get a generated entity from factory without knowing the creation of an entity also a advanced job class.

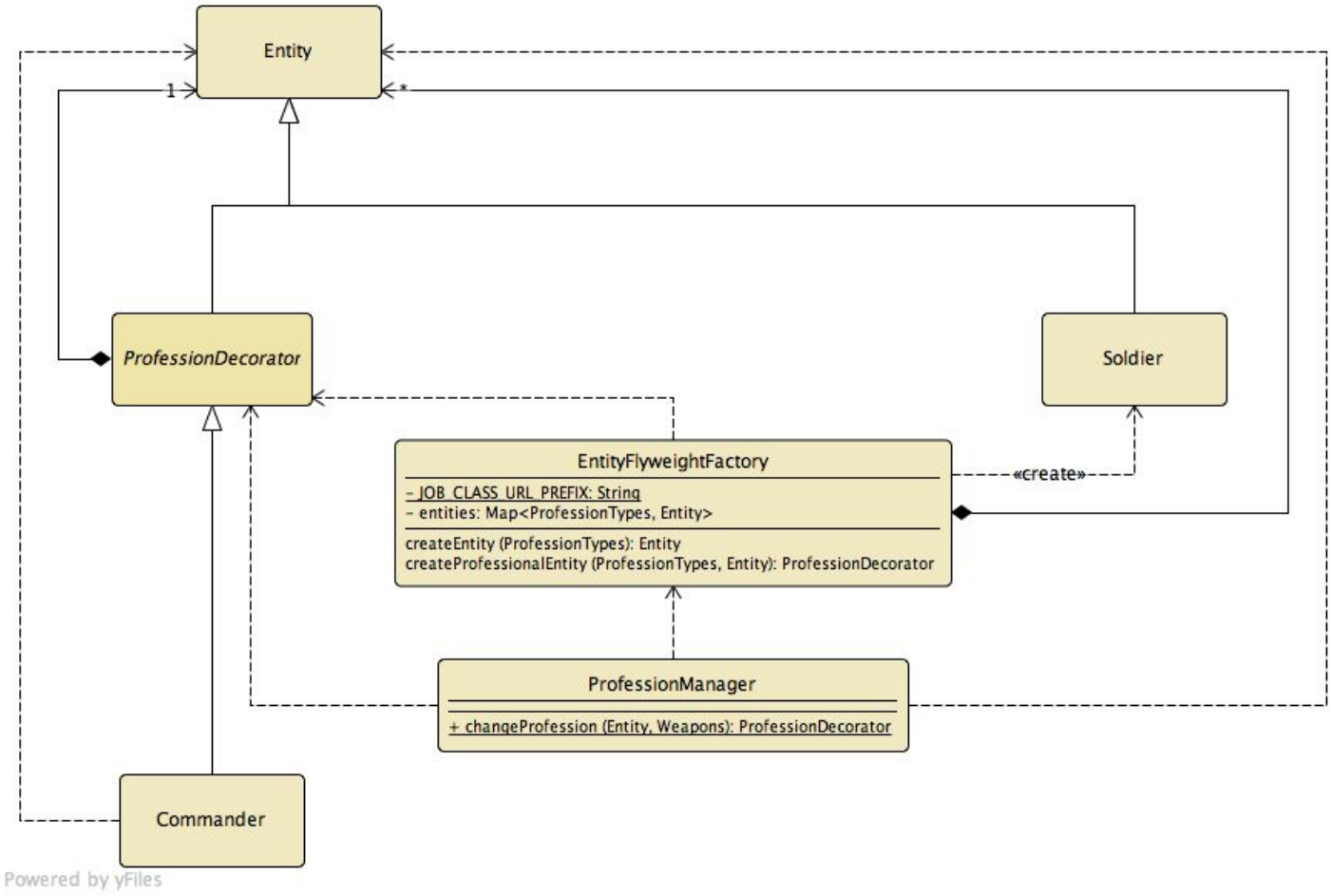
Structural

- **Decorator:**
- The decorator allows the developer modify an object dynamically, it is more flexible than inheritance, and simplifies code because you add functionality using many simple classes. In our Human VS. Alien Game, professionDecorator class have many subclass like Cheerleader, Area Attacker, chief, etc.
- In our assignment, advance professional has special ability that is different to basic attack function. Initially, all pieces in a team are basic unit "Soldier" or "Spawn". Basic unit dynamically upgrade to advanced professional class when picked up a specific weapon on the map. So what we need here includes:
  1. Basic invoke function is still needed for Entity class
  2. More abilities are able to attach to an entity dynamically

  In order to achieve it, Composition design pattern is implemented here. All advanced professionals extends ProfessionDecorator class and implement abstract method "Invoke(Entity target)" for getting its dedicate skill work.
- However, we do not really make best use of this pattern here. In the future we could apply it into job system to make one pieces has more than one skill when it pick up more weapons.

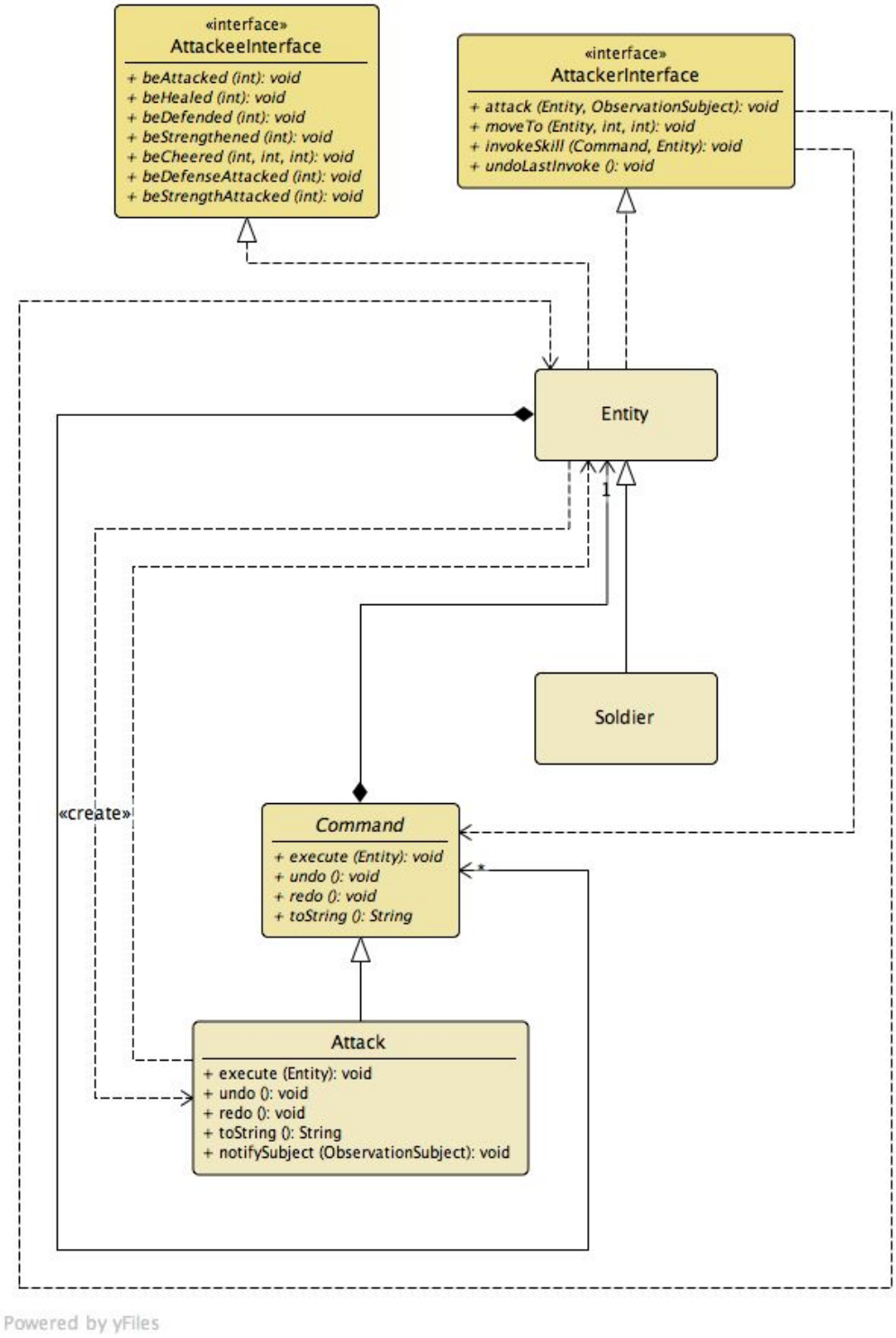


- **Flyweight:**
- The intent of this pattern is to use sharing to support a large number of objects. Flyweight design pattern to reduce memory usage about share Objects that are similar in some way rather than creating new ones. The client uses a Flyweight Factory to get a FlyWeight. The client change the profession job in the game, then call the EntityFlyweightFactory to get the CreateEntity and CreateProfessionalEntity, then to call the other method from Entity and ProfessionDecorator to Create the Profession positions.
- To handle the large amount of entity needing for our game if user decided to have lot of pieces on the board, this pattern will be very useful for reduce the memory usage.


Entity
1
*
ProfessionDecorator
Soldier
EntityFlyweightFactory
- JOB_CLASS_URL_PREFIX: String
- entities: Map<ProfessionTypes, Entity>
createEntity (ProfessionTypes): Entity
createProfessionalEntity (ProfessionTypes, Entity): ProfessionDecorator
«create»
ProfessionManager
+ changeProfession (Entity, Weapons): ProfessionDecorator
Commander
Powered by yFiles

Behavioural

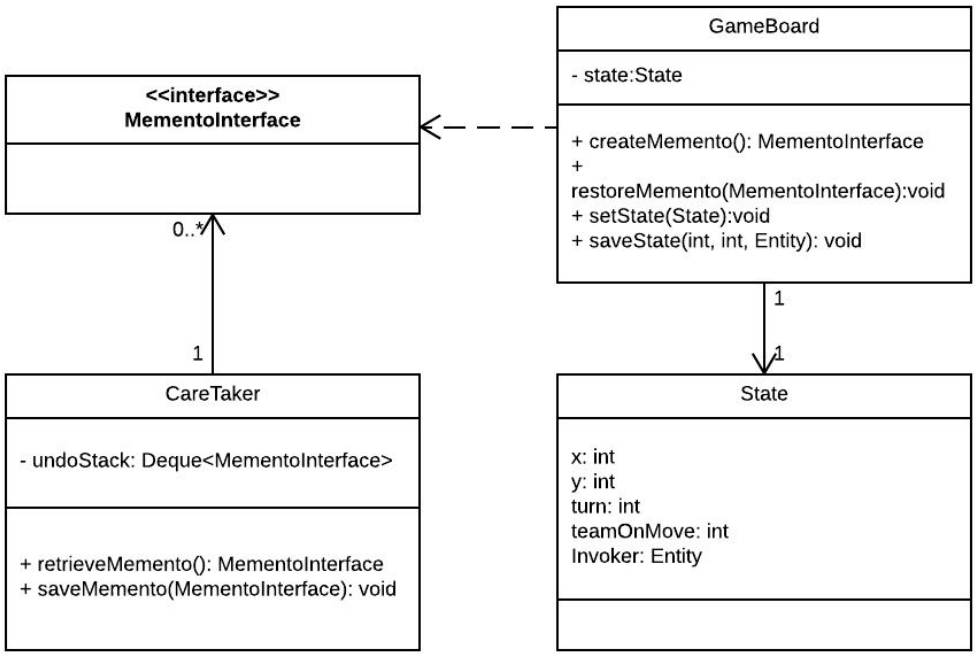
- **Command:**
- The client call the create attack function in entity, and entity call the attack class, and call execute and call command execute to execute the right code. Command design pattern allows developer to set aside a list of commands for later use.



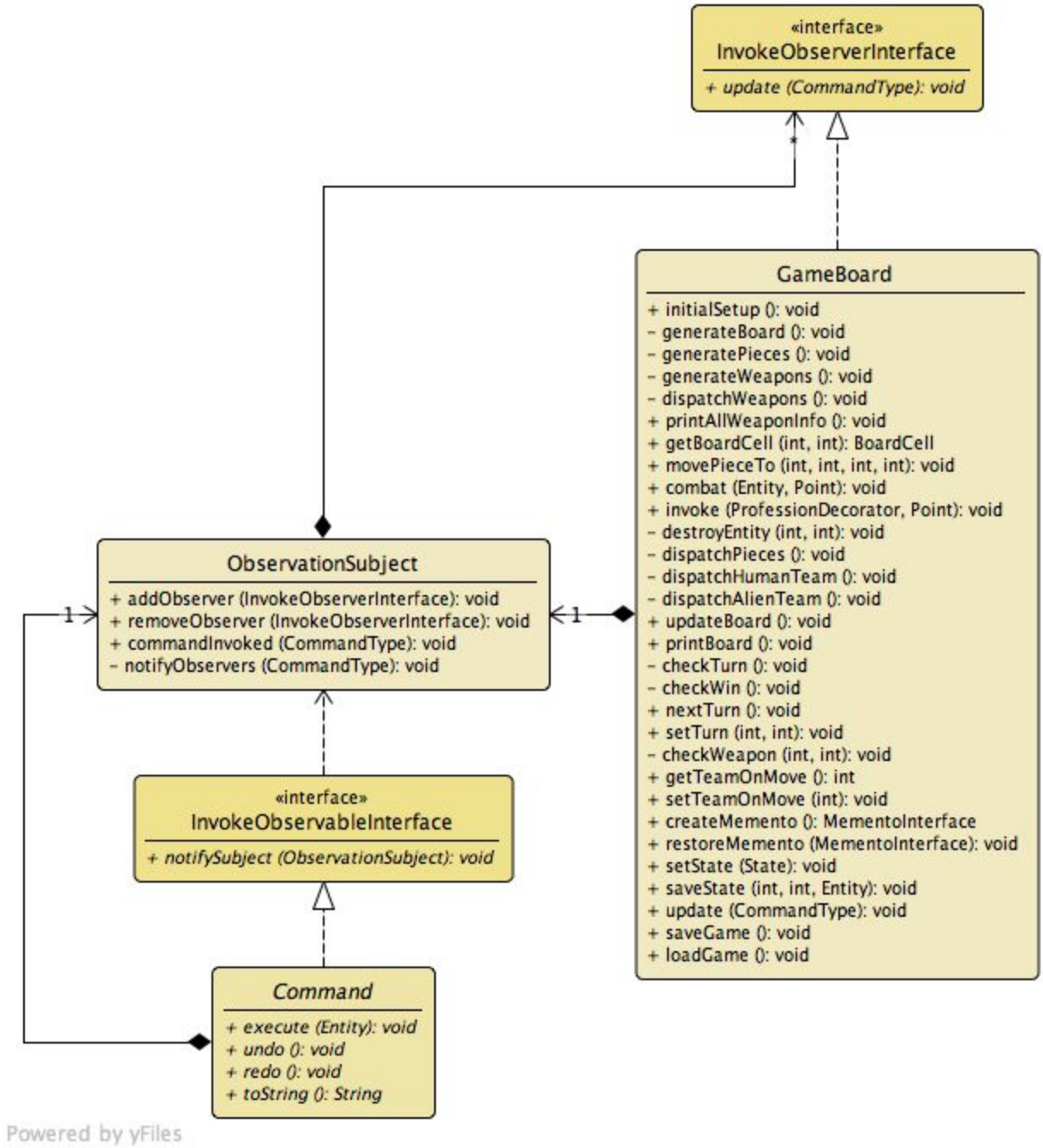
- To encapsulates all requests as objects to provides more functionality to a client.
  Client: Entity.class and its subclasses Command object can be created in client instance then

encapsulate parameters that needed for an command. It includes previous state of a target object and reference of target itself. When a command be created, it will be stored in a undo stack in the entity object. It can be use for undo function to retrieve previous status.

- **Memento:**
- Originator: GameBoard.class
- Memento: GameBoard -> Memento
- CareTaker: CareTaker.class
- Client: GameController.class
- Memento pattern provides a good solution to save snapshots of current game state and saves it as an encapsulated memento object that store in another individual class: caretaker. Memento object can only be access and decode by originator class.



- **Observer:**
- Command Object is created by a lower level object of Gameboard. Gameboard need to be notified when a command is performed. Then gameboard can pass a parameter to game controller for updating command animation on the view. It provides great solution to reduce the coupling and redundant code to handle dealing with getting notification from a command being executed. We do not have to implement different functions to handle choosing correct animation for different type of command.



Powered by yFiles