
COMP 62542 Pattern-Based Software Development
Group 5

Team Members

Product Owner	Salim Salim
Scrum Master	Carlos Valarezo
UX Expert	Olivier Staub
Architects	Bing Xu
	Dingxin Yu
Quality Assurance	Aurélie Pallas

Work Distribution

Pair-programming:

Creational Patterns:

Carlos

Dingxin

Structural Patterns:

Aurélie

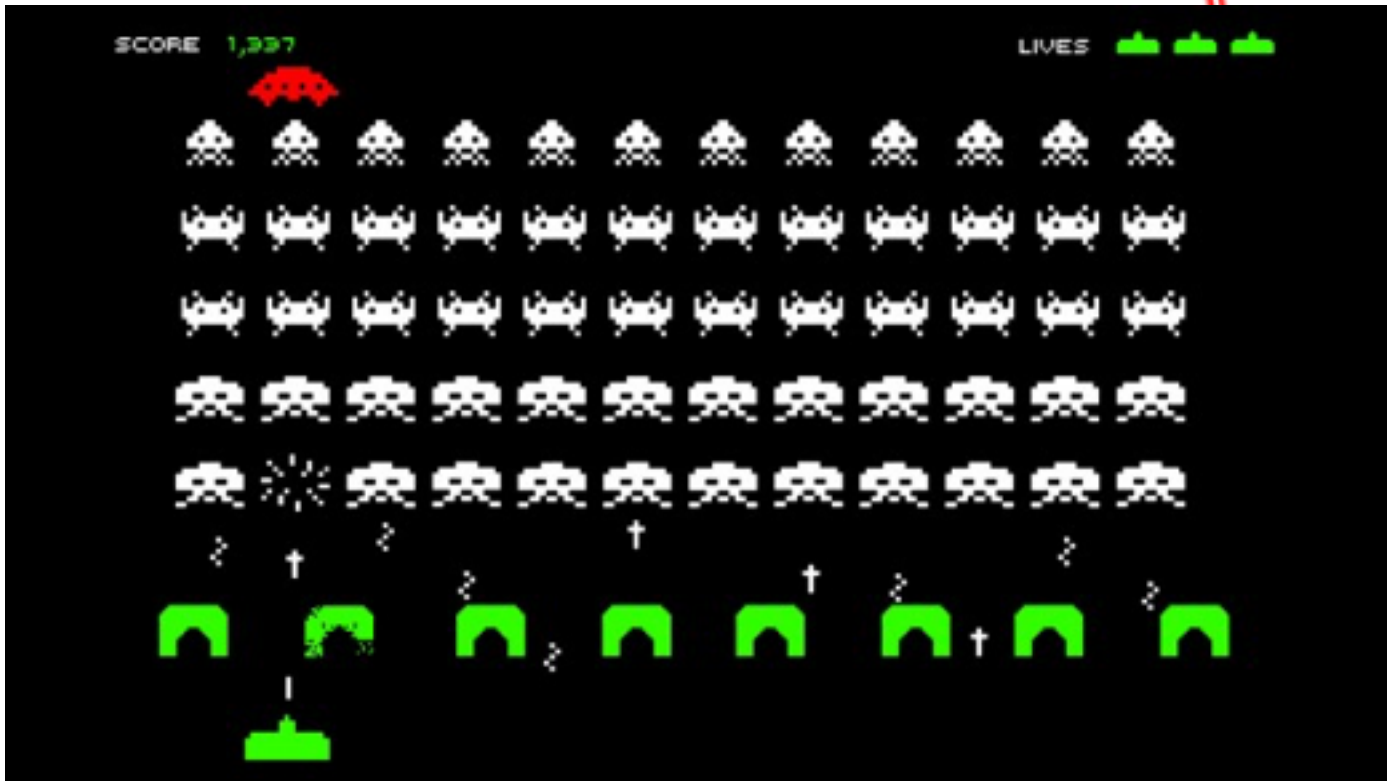
Bing

Behavioural Patterns:

Olivier

Salim

Space Invaders Game



Pattern Selection

Creational Patterns

- Singleton
- Abstract Factory

Structural Patterns

- Composite
- Flyweight

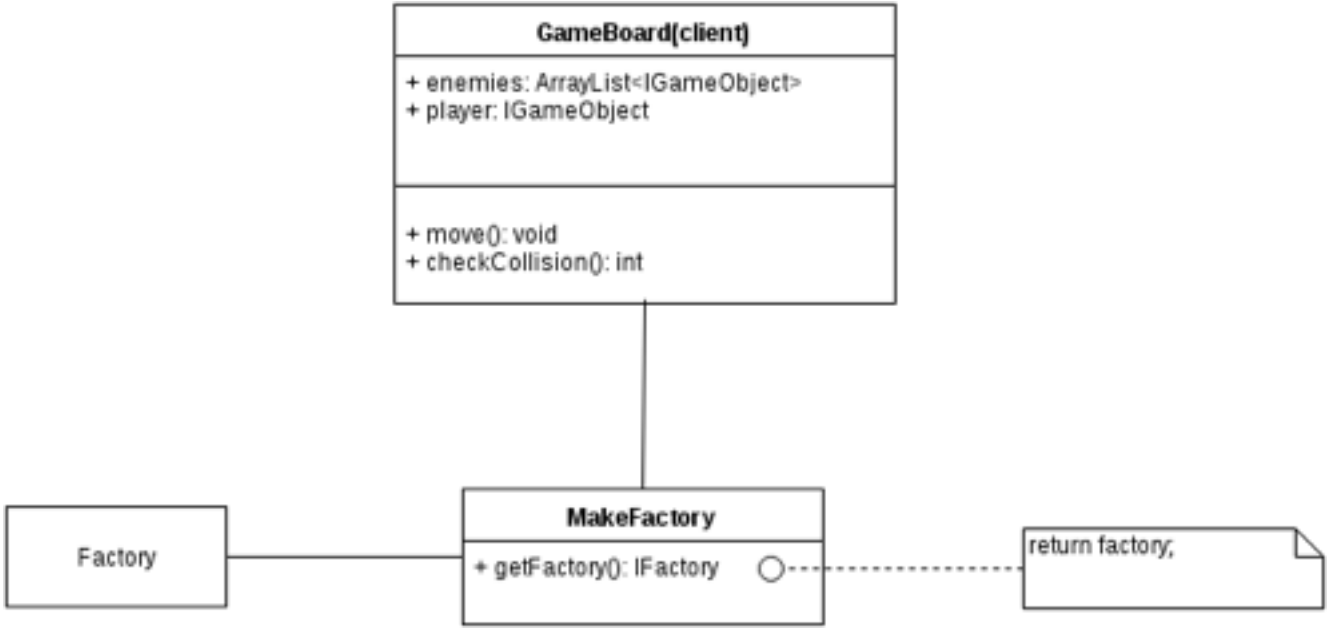
Behavioral Patterns

- State
- Strategy

Singleton Pattern

- Description:
  - Only one instance of a particular class
- Motivation:
  - Have only one instance of a Factory class
- Usage:
  - There is only one factory instance to create objects (client GameObject)
  - This instance can be used throughout the game

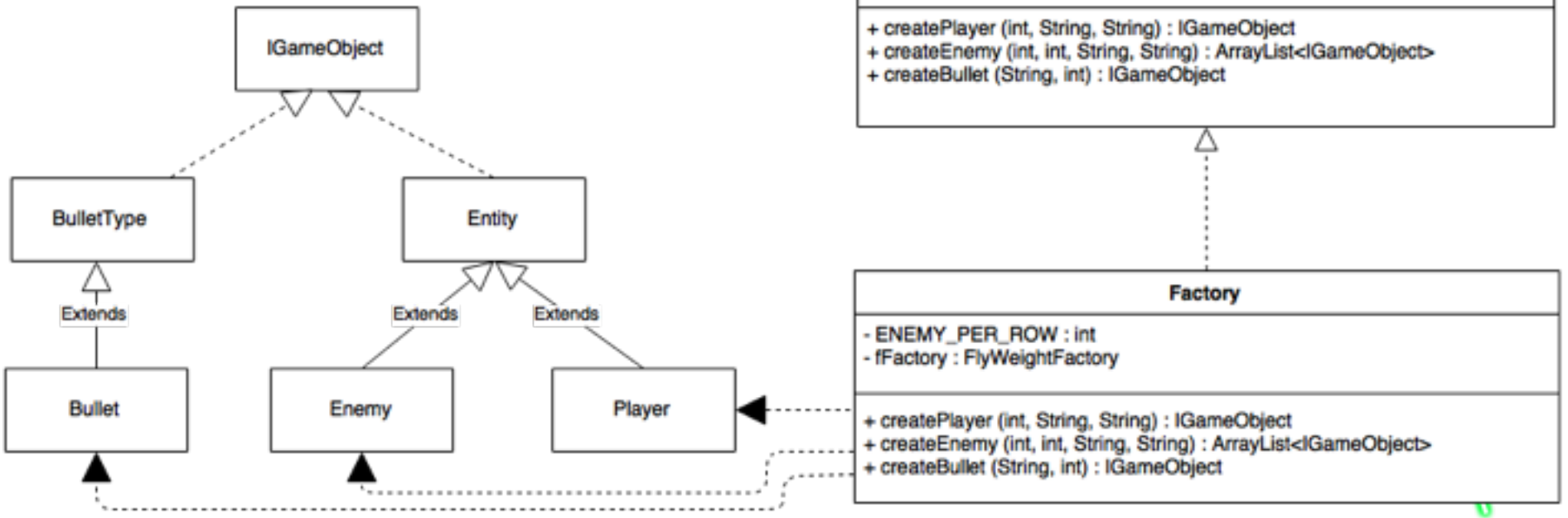
Singleton Diagram



Abstract Factory Pattern

- Description:
  - Create different kinds of object in different situations
- Motivation:
  - Only three types of objects
    - Player, Enemy & Bullet
  - Concrete factory classes for these objects are not necessary for this game
- Usage:
  - Create object instances when needed

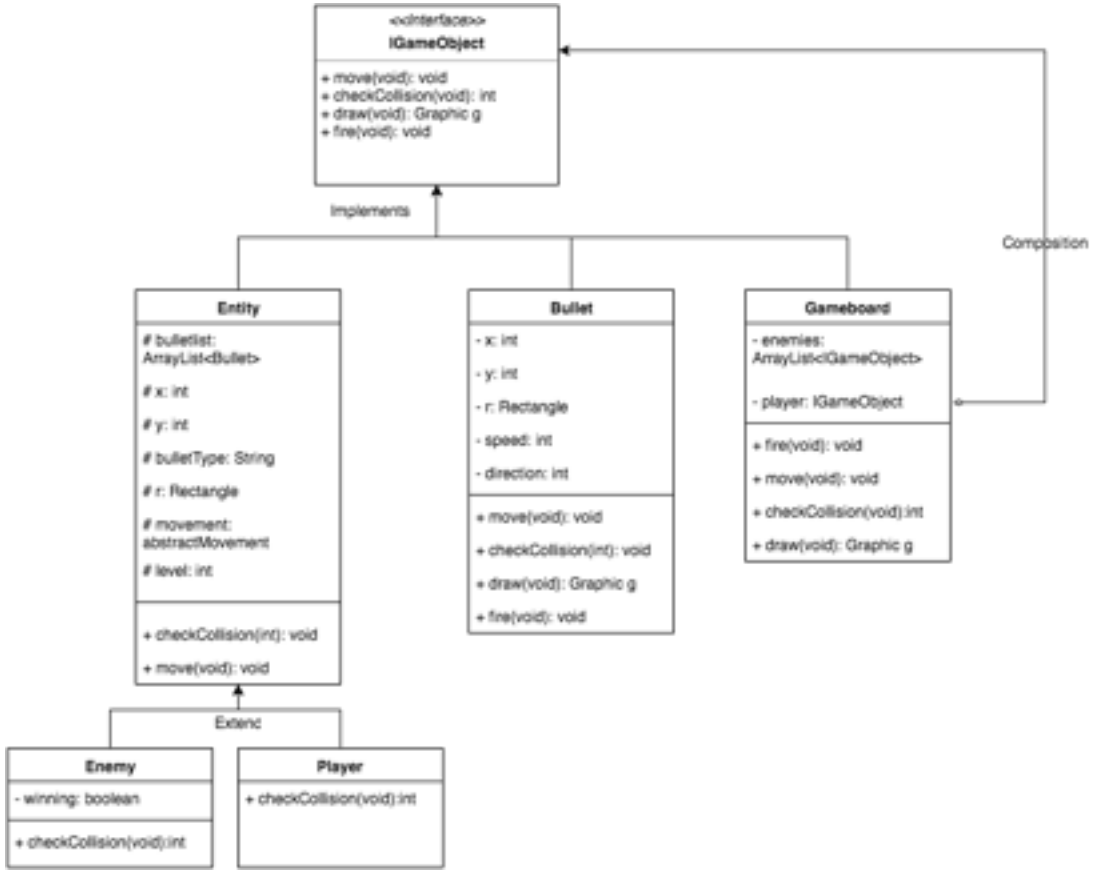
Abstract Factory - Class Diagram



Composite Pattern

- Description:
  - Treat a collection of objects as an individual
- Motivation:
  - We need a container(gameboard) to compose:
    - Entity
    - Bullet
- Usage:
  - Gameboard controls move, fire, checkcollision and draw for each IGameObjects

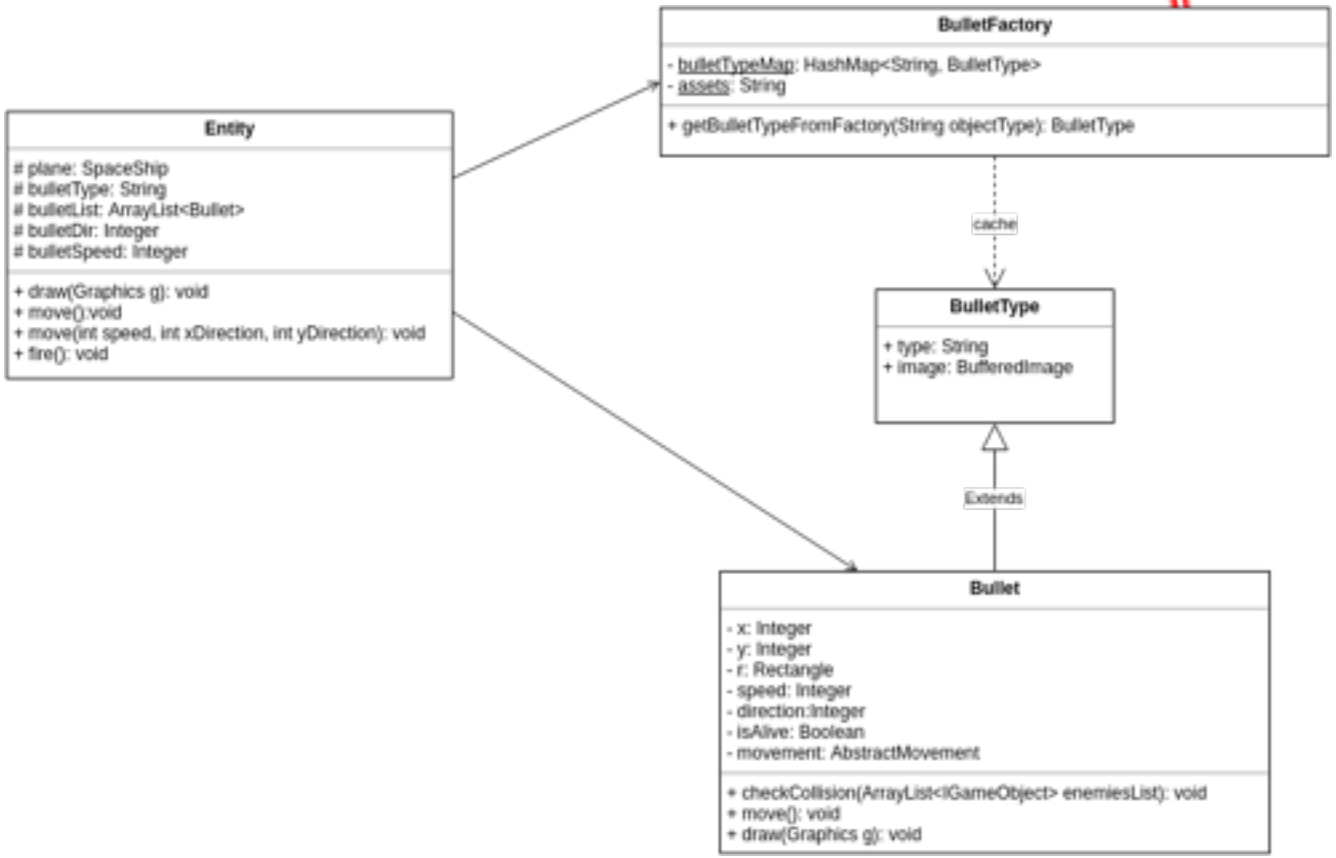
Composite Pattern - Class Diagram



Flyweight Pattern

- Description:
  - Shared object independent from the context
  - Separate a list of object between shared objects
  - We create the shared object only once
- Motivation:
  - Load each image only once
  - The creation of several enemies of the same type
- Usage:
  - One creation for each type of enemy and bullet

Flyweight Pattern - Class Diagram



State Pattern

Description:

Alter game behaviour

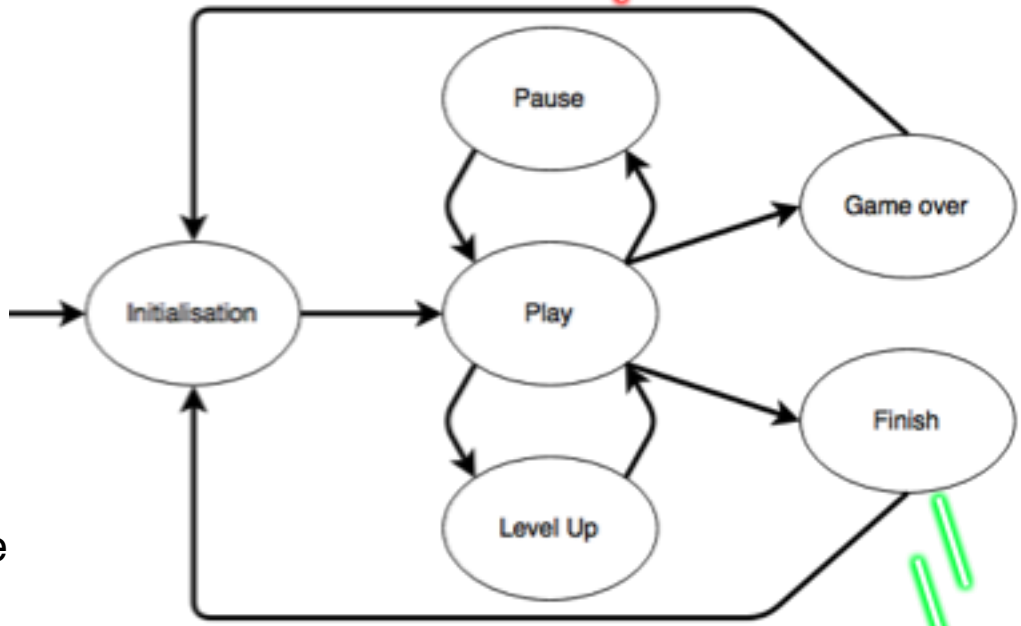
Motivation:

Game behaviour depends on its current state

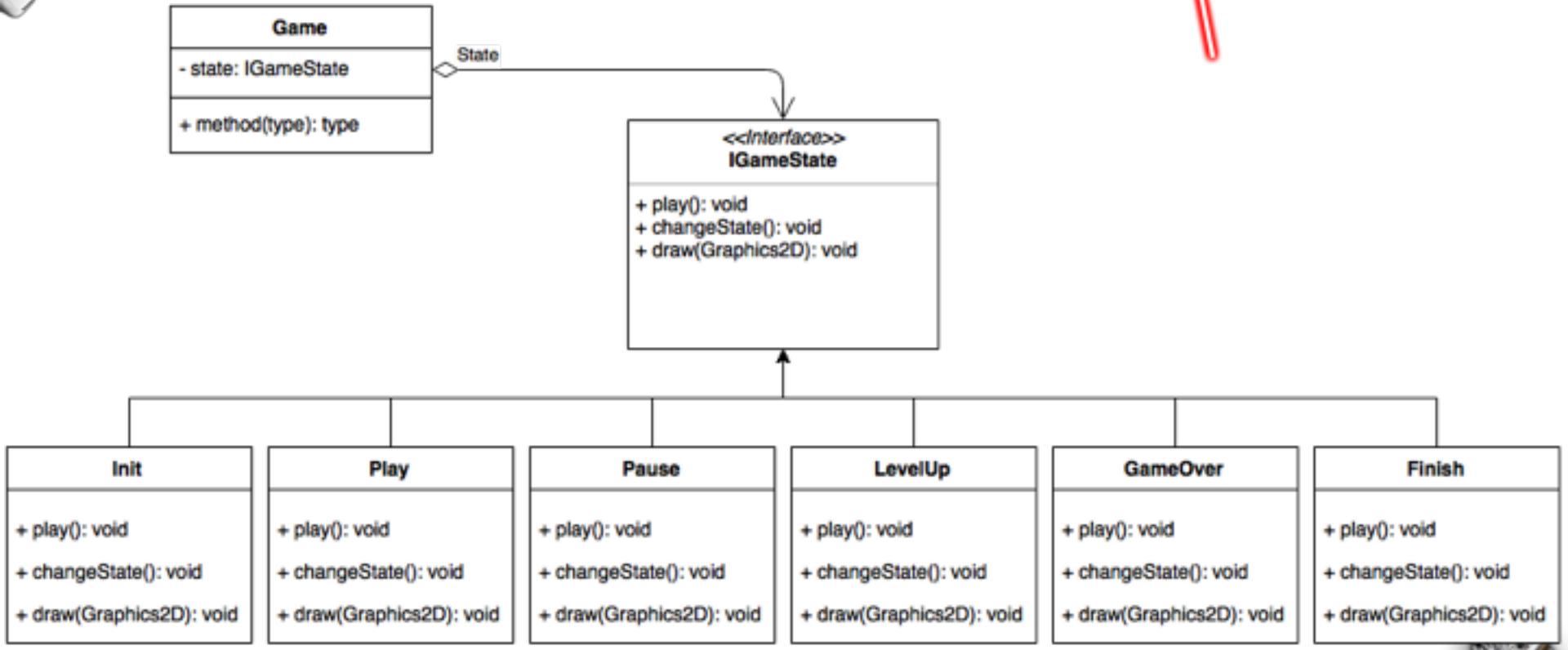
Usage:

Each phase of the game is a state

Each state implements its own behaviour



State Pattern - Class Diagram



Strategy Pattern

Description:

Encapsulate different movement algorithms

Motivation:

Different types of movement

Player depends on key press to move

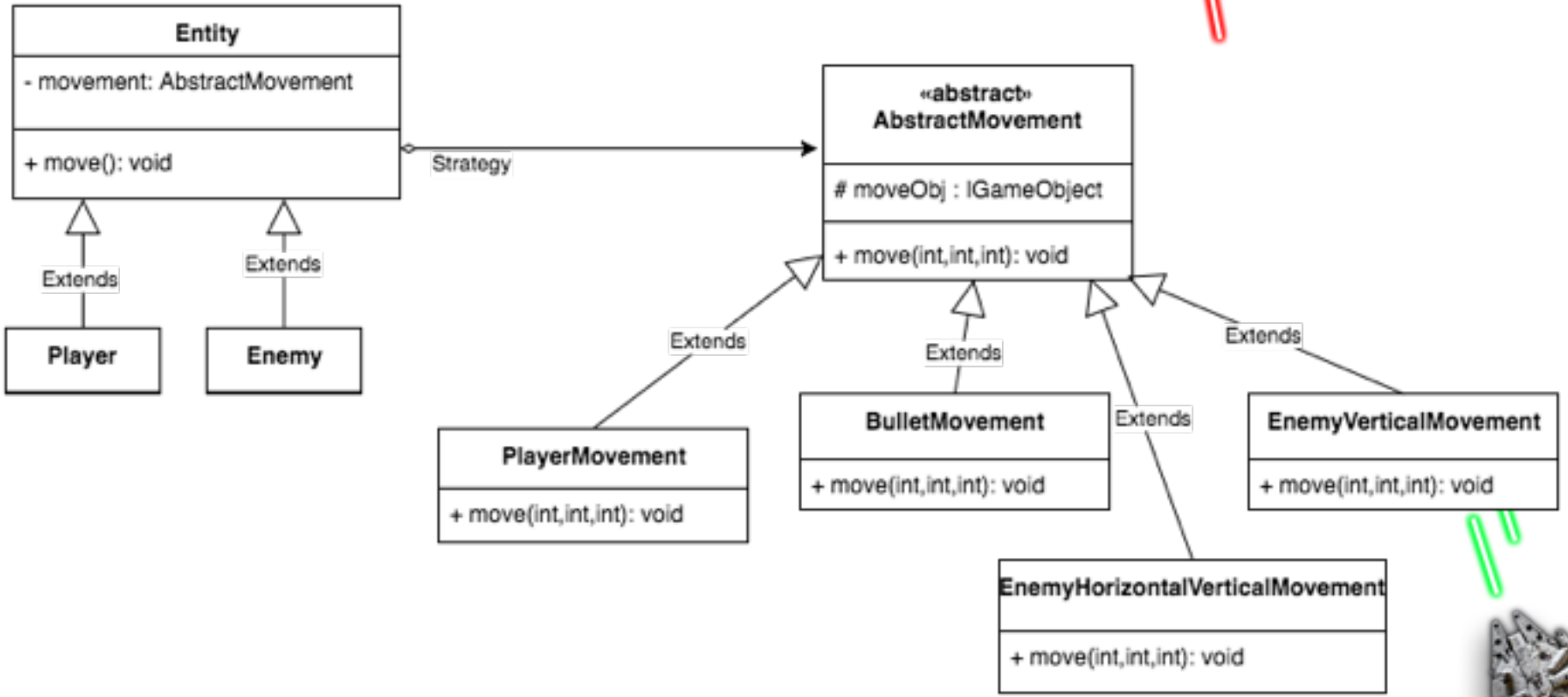
Enemy move differently in each level

Usage:

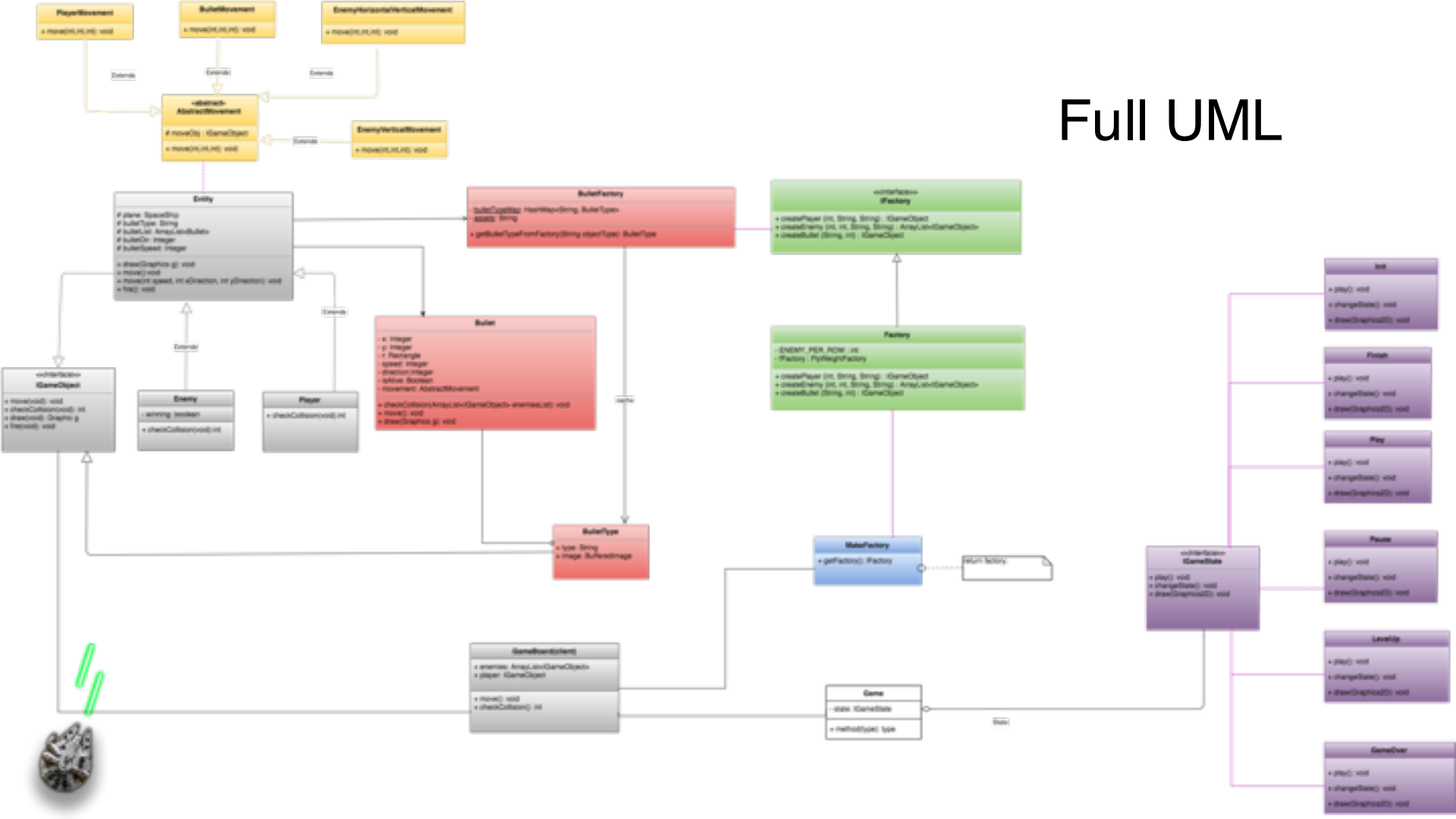
We separate movement types into multiple classes

Movement strategy is decided during entity creation

Strategy Pattern - Class Diagram



Full UML


PlayerMovement
BulletMovement
EnemyHorizontalVerticalMovement
Extends
«abstract»
AbstractMovement
EnemyVerticalMovement
Entity
BulletFactory
«interface»
IFactory
Bullet
Enemy
Player
«interface»
IGameObject
Factory
BulletType
MakeFactory
GameBoard(client)
Game
«interface»
IGameState
Init
Finish
Play
Pause
LevelUp
GameOver

Implementation Plan

Creational Patterns:

Singleton: A class to manage creation of factory

Abstract Factory: Only one concrete factory instance

Structural Patterns:

Composite: Inheritance

Flyweight: Inheritance

Behavioural Patterns:

State: Inheritance and Delegation

Strategy: Inheritance and Delegation

Challenges and Difficulties

Choosing patterns

Making them work together

Change of pattern design


May the patterns be with you