ITSP 2015-PROPOSAL

DESIGNING A FOOTBALL ENGINE

By- Rohan, Rupanshu, Sumith, Yogesh

1. ABSTRACT

We will design a football(soccer) game engine. The main objectives being

- To identify the entities in your game system.
- To identify the design problems.
- To apply patterns to address your design specifications.

2. MOTIVATION

Motivation being to simulate the backend of FIFA Manager(though the end product might not be that robust). En route we also get to learn about various data structures and algorithms and their implementation along with the concept of pattern centric software engineering.
The idea being that developers tend to reuse class relationships and object collaborations to simplify the design process. A design pattern consists of various cooperating objects (classes, relationships etc) - a consistent idiom for designers and programmers to speak about their design. Some examples being observer, decorator, strategy and builder patterns. A design pattern is normally represented with the help of a UML diagram which will also be learnt during the process.

3. IMPLEMENTATION

Let us assume that the end user is going to operate the game in the following sequence .

- Start the game
- Select two teams
- Add or remove players to form a team
- Pick a play ground
- Start the game

3.1. Identifying Entities

To list a few real world objects in the system, we have

- **Player** who plays the soccer.
- **Team** with various players in it.
- **Ball** which is handled by various players.

- **PlayGround** where the match takes place.
- **Referee** in the ground to control the game.

3.2. Identifying design problem

Here are a few design problems related to our real life entities:

- **Ball**
    - When the position of a ball changes, all the players and the referee should be notified straight away.
- **Team and Team Strategy**
    - When the game is in progress, the end user can change the strategy of his team (E.g., From Attack to Defend)
- **Player**
    - A player in a team should have additional responsibilities, like Forward, Defender etc, that can be assigned during the runtime.
- **PlayGround**
    - Each ground constitutes of gallery, ground surface, audience, etc - and each ground has a different appearance.(**Optional** UI implementation)

3.3. Identifying patterns to use

Addressing the design problems related with the 'Ball'

- ***Specific Design Problem: "****When the position of a ball changes, all the players and the referee should be notified straight away."*
- ***Problem Generalized:*** *"When a subject (in this case, the ball) changes, all its dependents (in this case, the players) are notified and updated automatically."*

Observer Pattern: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

- In this case, we used this pattern because we need to notify all the players, when the position of the ball is changed.

Addressing the design problems related with 'Team' And 'TeamStrategy'

- ***Specific Design Problem:*** *"When the game is in progress, the end user can change the strategy of his team (E.g., From Attack to Defend)"*

- ***Problem Generalized: "****We need to let the algorithm (TeamStrategy) vary independently from clients (in this case, the Team) that use it."*

Strategy Pattern: Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Addressing the design problems related with 'Player'

In our case, sub classing (inheritance) is not the suitable method, because we need to separate the responsibilities like 'Forward', 'Midfielder', 'Defender' etc from the player implementation. Because, a player can be a 'Forward' one time, and some other time, the same player can be a 'Midfielder'.

- ***Specific Design Problem:*** *"A player in a team should have additional responsibilities, like Forward, Defender etc, that can be assigned during the runtime."*
- ***Problem Generalized: "****We need to attach additional responsibilities (like Forward, Midfielder etc) to the object (In this case, the Player) dynamically, without using subclassing"*
- We can chose the 'Decorator' pattern to address the above design problem.

Decorator Pattern: Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

Addressing the design problems related with 'PlayGround'

- ***Specific Design Problem:*** *"Each ground constitutes of gallery, ground surface, audience, etc - and each ground has a different appearance."*
- ***Problem Generalized:*** *"We need to separate the construction of an object (ground) from its representation (the appearance of the ground) and we need to use the same construction process to create different representations."*

Builder Pattern: Separate the construction of a complex object from its representation so that the same construction process can create different representations.

4. LEARNING OBJECTIVE

The main aim of the project will be to learn the following :

- Learn about design patterns(observer, decorator, strategy and builder Patterns) and concept of pattern centric software engineering.
- Reading and understanding UML diagrams.
- You will identify the design problems in your football game engine and deciding on the design.

- Read and implement the various data structures and algorithms.
- Maintaining a well oiled project with fixed code design, simultaneous documentation and good testing environment.

5. DEMONSTRATION

At the end of this project we hope to give a working Football Engine , in which the user can

- Start the game
- Select two teams
- Add or remove players to and from a team
- Pick a play ground
- Start the game

Also change the strategies, substitution etc in runtime.

6. TIMELINE

- ➔ Till May 5
  - ◆ Follow up on proposal
  - ◆ Reading up on patterns and getting familiar with UML diagrams
- ➔ Week 1 (May 5- May 12)
  - ◆ Implementation of ‘Ball’.
  - ◆ Learning and implementation of Observer Pattern.
- ➔ Week 2 (May 12- May 19)
  - ◆ Implementation of ‘Team’ and ‘TeamStrategy’.
  - ◆ Learning and implementation of Strategy Pattern.
- ➔ Week 3 (May 19- May 26)
  - ◆ Implementation of ‘Player’.
  - ◆ Learning and implementation of Decorator Pattern.
- ➔ Week 4 (May 26- June 2)
  - ◆ Implementation of ‘Playground’
  - ◆ Learning and implementation of Builder Pattern
- ➔ Week 5 (June 2 - June 9)
  - ◆ Assembling the engine using the above implemented APIs.
  - ◆ Feed in data of different teams and players
- ➔ Week 6 (June 9- June 16)
  - ◆ Buffer week
  - ◆ Working on documentation
  - ◆ Unfinished details/To Dos