Mobile App Development Using Software Design Patterns

Nicole Barakat, Doan Nguyen
California State University, Sacramento 95819

I. Introduction

Computer Science (CSC) 133 course is an Object-Oriented Computer Graphics Programming class offered at California State University, Sacramento. This poster paper describes a class project utilized in Spring 2018. The course materials where this assignment was based from the pedagogical study [2] that redesigned CSC 133 by utilizing CodenameOne (CN1) [1], a cross-platform mobile application development environment, to teach topics of this course that originally utilized Java Standard Edition and desktop application development. With the purpose of incorporating mobile application development experience to the existing course, this pedagogical study updated the course content and modified the original course materials [3] (e.g., slides, projects, and code samples) developed mainly by colleagues who previously taught CSC 133. The specification and design of this class project in Spring 2018 (i.e., Asteroid game) are also originated from the original CSC 133 course materials. The work here converted the Asteroid game from Java into CN1 by using the converted project (Race Car game) from [2] as its model. The most challenging aspect was to find the proper components in CN1 which can perform the comparable functions and at the same time preserving the core design of the system [3].

As inferred by the class title, this class focuses on the fundamental concepts of the object-oriented (OO) paradigm, introduction to Computer Graphics, as well as Mobile App Development. The object-oriented paradigm encompasses many pertinent topics such as: polymorphism, inheritance, encapsulation, and abstraction. These concepts are supported through formalisms such as UML diagrams and software design patterns. Emphasis is put on implementing event-driven systems through the study of computer graphics.

These concepts are all tied together through the construction of an enticing cross-platform mobile application. CN1 is the mobile app development environment tool chosen due to its Java focus and cross-platform capabilities. The Java programming language works exceptionally well for the purpose of this course, being that it is an object-oriented, ubiquitous, and versatile language. Many applications are limited to Android/IOS, which makes this deployment even more salable and overall appealing. Throughout the semester, the students are taught how to develop a mobile application given the knowledge of efficient structuring principles and tools. Incentive for hard work is encouraged by the semester end goal, the creation of an "Asteroid Game." This paper is organized into three parts. The first part describes the design aspects of the application. The second part highlights the testing areas. The paper ends with remarked conclusion and future work.

II. Design

A specification is given for the assignment [2,3], requirements are analyzed, then a UML class diagram is created. This diagram gives an excellent visualization of the organization and basic structure needed. It shows many of the important relationships and concepts needed to structure and execute an effective system. The underlying structure of this application is largely inspired by, and incorporates, many software design patterns including an emphasis on the Model View Controller (MVC) architecture. The MVC architecture allows for the organization and compartmentalization of code into three main modules, which in turn largely contributes to scalability and maintainability. In the context of this course, an interactive game application is created with the following modules: the Game class is the user's "controller", the GameWorld class is the "model" that holds the majority of the complex data which is accessed and manipulated by the controller, and the PointsView and MapView are the views through which the game state may be conveyed to the user. Figure 1 shows a condensed UML class diagram of the overall system architecture.

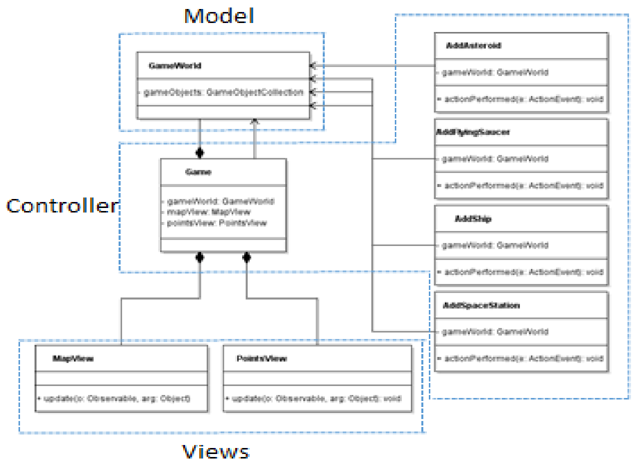

Model
GameWorld
- gameObjects: GameObjectCollection
AddAsteroid
- gameWorld: GameWorld
+ actionPerformed(e: ActionEvent): void
AddFlyingSaucer
- gameWorld: GameWorld
+ actionPerformed(e: ActionEvent): void
Controller
Game
- gameWorld: GameWorld
- mapView: MapView
- pointsView: PointsView
AddShip
- gameWorld: GameWorld
+ actionPerformed(e: ActionEvent): void
AddSpaceStation
- gameWorld: GameWorld
+ actionPerformed(e: ActionEvent): void
MapView
+ update(o: Observable, arg: Object)
PointsView
+ update(o: Observable, arg: Object): void
Views

Figure 1. Condensed UML of the game's overall architecture

This application is further coordinated using various creational, structural, and behavioral design patterns such as the Proxy, Composite, Command, and Observer/Observable patterns.

10.18293/SEKE2018-324

Proxy Design Pattern: The Proxy structural pattern is used as a protection proxy to protect the game's state by prohibiting the views from modifying the game world. This structure defines an interface which strictly specifies what methods and actions a client program may perform. For example, IGameWorld contains the restricted methods which will be implemented by both the real GameWorld class, as well as the GameWorldProxy class.

Composite Design Pattern: The objects of the "Asteroid Game" are organized in a hierarchical manner: a GameObject can be a FixedObject or a MoveableObject, a MoveableObject can then be further divided into: Ship, Asteroid, FlyingSaucer, or Missile. As can be seen, some objects, such as MoveableObject are groups of other objects. Additionally, these moveable objects, which are all moveable, are treated uniformly through inheritance and implementation of an IMoveable interface.

Command Design Pattern: A user has a multitude of commands available upon beginning the game. Maintenance of state information of each command is done by encapsulating the various commands that the player may invoke. The constructor of the main Game class creates strictly one instance of each command object, and reuses that object as needed. Despite the origin of invocation of the command, the execution of a code block is the same. This strategy, therefore, avoids the need for multiple copies of code that perform the same task.

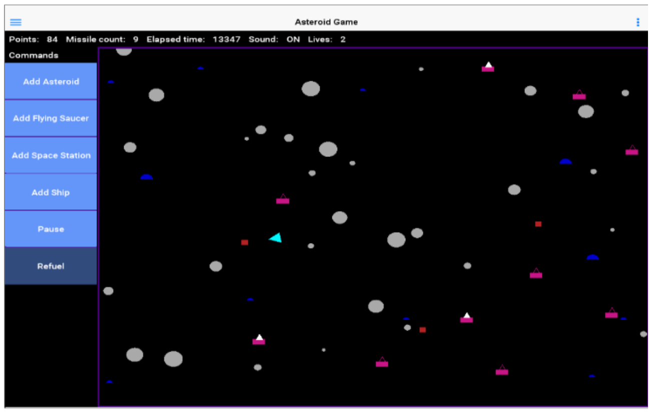
Observer/Observable Design Pattern: The MapView and PointsView classes are registered as observers of the observable, GameWorld. Whenever any changes are made that would affect the state of the views, GameWorld notifies its observers of updates. This relationship allows for proper coordination and communication between participating classes when changes occur.

Animation, Objects Interaction, Sound

A system timer's tick function is generated. For each tick, an object movement is computed using its current positions, headings, and speeds. Collisions are detected and handled polymorphically through the use of an ICollider interface containing collidesWith and handleCollision methods which are implemented by all objects that have the capability of colliding. The collidesWith method checks whether or not two given objects are colliding or not by using either a 2D bounding circle. If a collision is confirmed, it must be handled appropriately in the handleCollision method; for an example, if a missile hits an asteroid, both objects must be deleted from the world and the user score increases.

III. **Testing**:

The verification and validation of this program is done using the test cases derived from the system specification [2,3]. We are focusing on black box testing. Additionally, a few model based models were also using to verify different stages of the game. For an example a sequence of the following actions: pressing pause, turning off sound, and pressing play, should yield a result of the sound not coming back on. A sample of Asteroid Game GUI is shown in Figure 2.


Asteroid Game

Points: 84 Missile count: 9 Elapsed time: 13347 Sound: ON Lives: 2

Figure 2. A sample of the Asteroid Game running under a CN1 simulator.

IV. **Conclusion**:

A project such as this, provides invaluable knowledge and skills to participating students, consequently building them up for future success. The production of a game showcases a rich understanding of vital concepts needed for being prosperous in the Computer Science field. No matter what area of expertise one pursues, the lessons learned in a class like this aids in a stronger resume, larger repertoire, and well-rounded abilities.

References

[1] CodenameOne Guide: Build Cross Platform Apps using Java, http://codenameone.com/manual/index.html, Accessed May 2018.

[2] Pinar Muyan-Ozcelik, A Hands-on Cross-Platform Mobile Programming Approach to Teaching OOP Concepts and Design Pattern, In Proceedings of Software Engineering Curricula for Millennials (SECM) Workshop at ACM/IEEE 39th International Conference on Software Engineering (ICSE), May 20-28, 2017, Buenos Aires, Argentina, doi: 10.1109/SECM.2017.12.

[3] J. Clevenger and S. Gordon, "CSC 133 - Object-oriented Computer Graphics Programming Lecture Notes and Assignments," Spring, 2014, California State University, Sacramento.