**TERM PROJECT**

Michael A Crowell

CSC 341 Software Design

Carroll University

April 24, 2012

**Introduction**

This program is a single player video game. The intent of the game is to provide simple, fast paced entertainment. The application is designed to be easy to understand and quick to play so the user can play with minimal demands on time.

In this game the user will move a rabbit around a play field, using the keyboard to input the controls that move the rabbit. Five keys are used during game play. The four arrow keys will move the rabbit in the indicated directions. The space bar is used to increase the speed of the rabbit as it moves. The object of the game is to catch the carrots and watermelons as they randomly appear at various spots on the play field. One object will appear at a time. Once one object is caught then a new object will appear at a new location. Trees are placed on the play field to provide obstacles to the player’s movement. Also providing a challenge to game play are four computer controlled animals that will chase the rabbit and attempt to eat the carrots and watermelons. Points are awarded each time the player successfully catches a carrot or watermelon. Points are deducted when one of the computer controlled animals catches the rabbit or one of the carrots or watermelons. Game play ends when a thirty second timer counts down to zero.

The application was designed to be flexible. Parameters can be easily changed with minimal code affected. The following documents describe and explain the designs that were implemented in this application.

**Software Analysis – Use Cases**

**Epic:**

As a user, I can play a simple game where I attempt to catch objects while I attempt to avoid getting caught. The objects that I will attempt to catch appear one at a time at random locations around the screen. Once an object is caught, points are awarded, and a new object appears at a new random location. Computer controlled characters attempt to catch the objects as well as the user controlled character. Points are awarded for each successful object that the user captures. Points are lost each time a computer controlled character captures an object or the user controlled character. The game operates on a preset timer. The timer counts down as game play progresses and game play ends once the timer expires.

**User Story**:

The end user is a game player who would like to see a game that is simple and fast paced. The need is for game controls that are easy to remember and game play that is easy to understand. The game should be able to be completed in a short time frame. The benefit is to provide the user with brief entertainment that can be enjoyed as a quick escape from other activities without requiring a major interruption in terms of time required to learn or play the game.

**Use Case: Summary**

*Use Case:* **User plays a single player video game.**

**Scope**: Single game locally on the user’s machine.

**Level**: Summary

**Intention in Context**: The intention of the User is to move a character around the screen, scoring points by capturing objects, and sustaining game play by avoiding getting caught by computer generated characters.

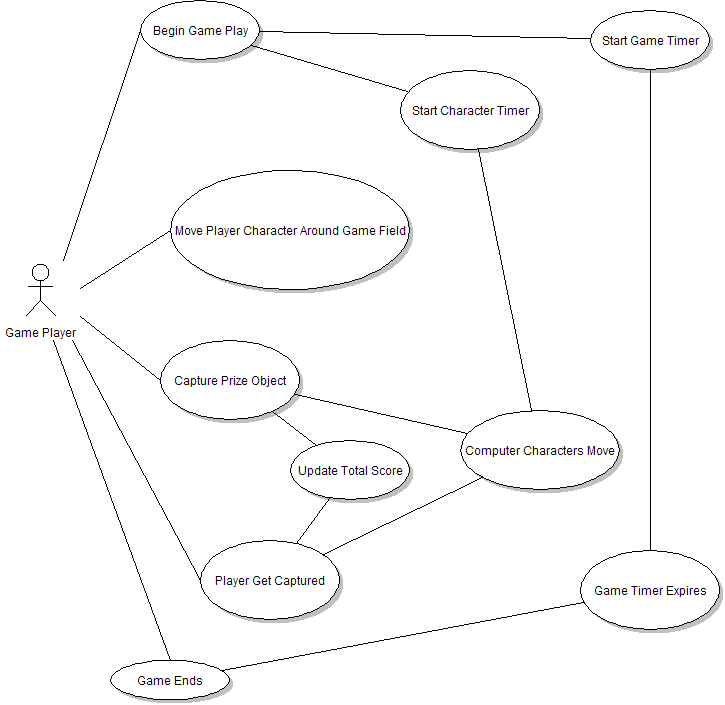
**Primary Actor**: Game Player

**Main Success Scenario**:

1. User begins game play.
2. Timer begins.

Steps 3 through 6 can be repeated according to the results of user movement.

1. User presses keys to move around screen
2. User captures an object and is awarded points.
3. Computer character captures object, user loses points.
4. Computer character captures user, user loses points.
5. Timer expires, game play ends.



**Use Cases: User Goal**

*Use Case:* **User begins game play.**

**Scope**: Single game locally on the user’s machine.

**Level**: User Goal

**Intention in Context**: The intention of the User is to start the video game and begin to move the user operated character around the screen. The user interacts directly with the system via keyboard input.

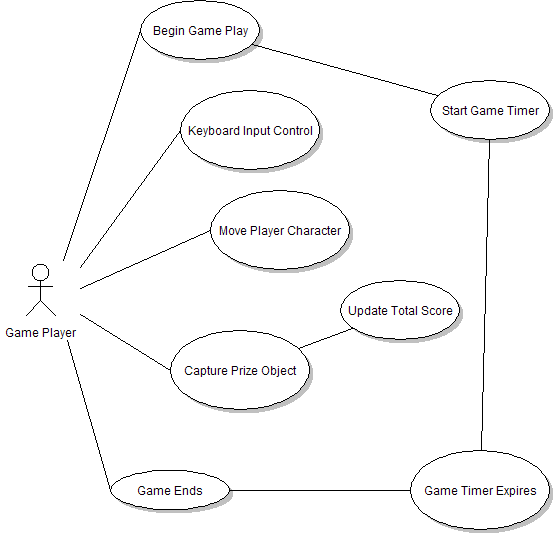
**Primary Actor**: Game Player

**Main Success Scenario**:

1. User begins game play.
2. Timer begins.

Steps 3 through 6 can be repeated according to the results of user movement.

1. User presses keys to move around screen
2. User avoids being captured
3. User captures an object and is awarded points.
4. Timer expires, game play ends.



*Use Case:* **User plays a round of the game.**

**Scope**: Single game locally on the user’s machine.

**Level**: User Goal

**Intention in Context**: The intention of the User is to continually move the user controlled character around the game field, using the keyboard as the mode to input the directions and speed of movement. The user will attempt to avoid objects on the screen and avoid being captured while trying to capture random prizes that appear on the screen.

**Primary Actor**: Game Player

**Main Success Scenario**:

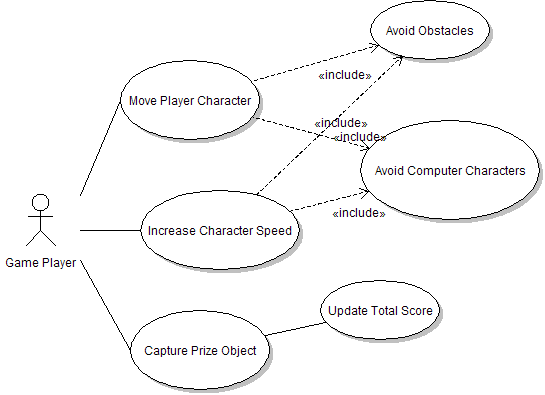
Steps 1 through 5 can be repeated according to the results of user movement.

1. User presses keys to move around screen.
2. User presses arrow keys on keyboard to input desired direction.
3. User presses space bar to increase speed.
4. User moves player character around obstacles on the screen.
5. User captures an object and is awarded points.

**Extensions**:

5a. User is captured by a computer controlled character. User loses points.

5b. The prize object is captured by a computer controlled character. User loses points.



*Use Case:* **User completes the video game.**

**Scope**: Single game locally on the user’s machine.

**Level**: User Goal

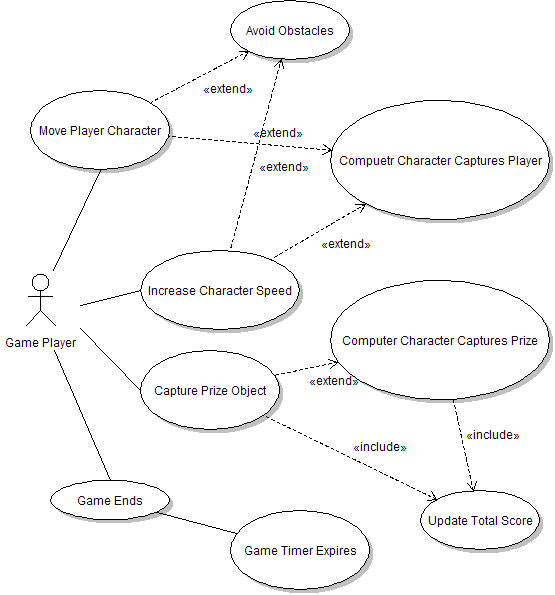
**Intention in Context**: The intention of the User is to play the video game through to its completion. Game play ends when the game timer expires. During game play the user is attempting to accumulate as many points as possible to achieve a high score at completion of the game.

**Primary Actor**: Game Player

**Main Success Scenario**:

Steps 1 through 7 can be repeated according to the results of user movement.

1. User presses keys to move around screen.
2. User presses arrow keys on keyboard to input desired direction.
3. User presses space bar to increase speed.
4. User moves player character around obstacles on the screen.
5. User captures an object and is awarded points.
6. Computer character captures object, user loser points.
7. Computer character captures user’s character, user loses points.
8. Timer expires, game play ends.



**Use Cases: Sub-Function**

*Use Case:* **Move the player character.**

**Scope**: Single game locally on the user’s machine.

**Level**: Sub-Function

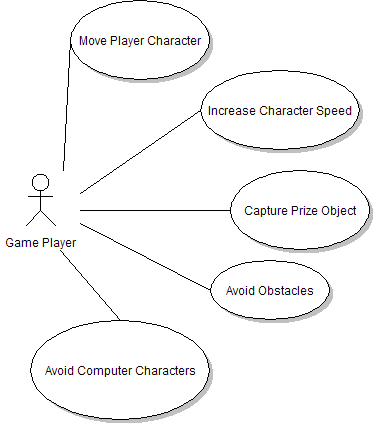
**Intention in Context**: The intention of the User is to move the player controlled character around the game field in an attempt to capture the objects that appear at random locations around the screen. The user presses one of the four arrow keys on the keyboard, up, down, left, or right, to move the on screen character in that particular direction. Additionally, pressing two arrow keys at the same time can cause the on screen character to move in a diagonal direction. The on screen character by default will move at a predetermined speed. The user can cause the on screen character to move at a faster speed by pressing and holding down the space bar while also pressing the arrow keys.

**Primary Actor**: Game Player

**Main Success Scenario**:

1. User presses one of the arrow keys, up, down, left, or right, the player controlled character moves in the direction of the arrow key that was pushed.
2. User presses and holds down the space bar while pressing the arrow keys, the speed of the player controlled character increases.

*Use Case:* **Capture an object.**



**Scope**: Single game locally on the user’s machine.

**Level**: Sub-Function

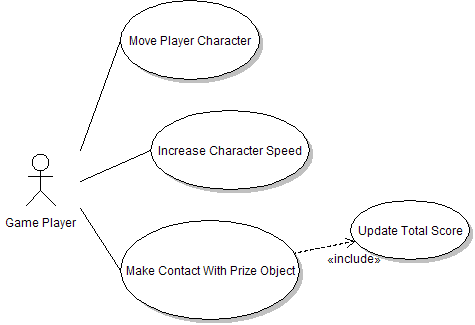
**Intention in Context**: The intention of the User is to capture a prize object with the on screen player controlled character. An object is captured when the player icon makes contact with the prize icon. The user presses the arrow keys and space bar to move the player character on the game field. When the player character makes contact with the prize object, points are added to the player total score, a new prize object appears at a new randomly generated location on the game field.

**Primary Actor**: Game Player

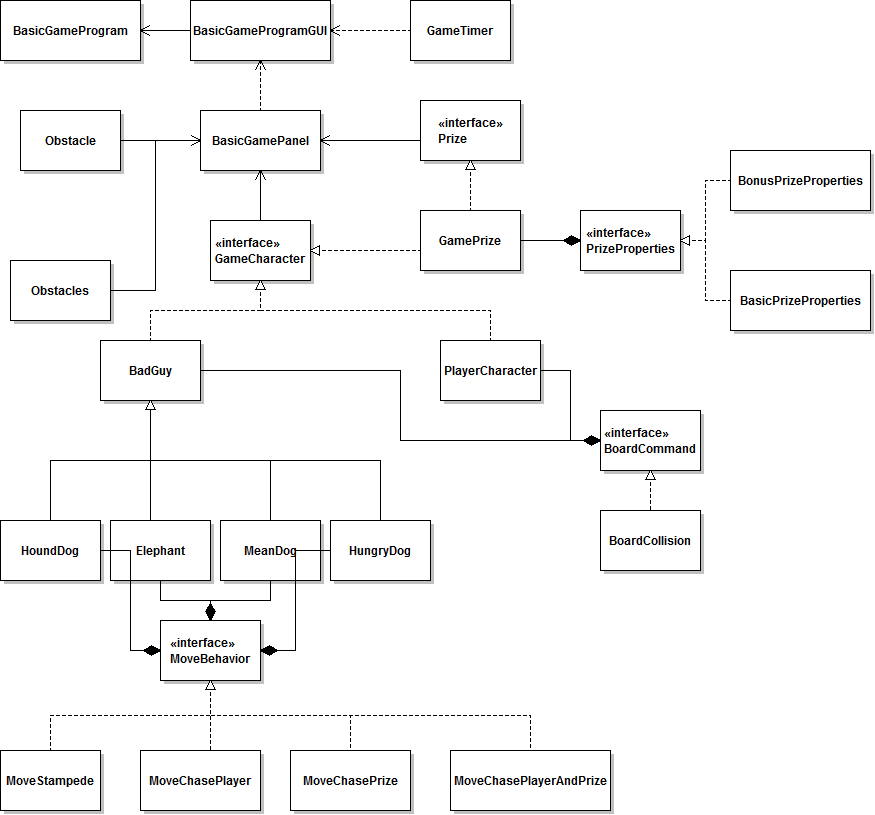
**Main Success Scenario**:

1. User presses one of the arrow keys, up, down, left, or right, the player controlled character moves in the direction of the arrow key that was pushed.
2. User presses and holds down the space bar while pressing the arrow keys, the speed of the player controlled character increases.
3. User directs the player controlled character toward the prize object, avoiding obstacles and computer controlled characters along the way.
4. The player controlled character makes contact with the prize object. Points are awarded to the player’s total score.

**Software Analysis – Other Diagrams**



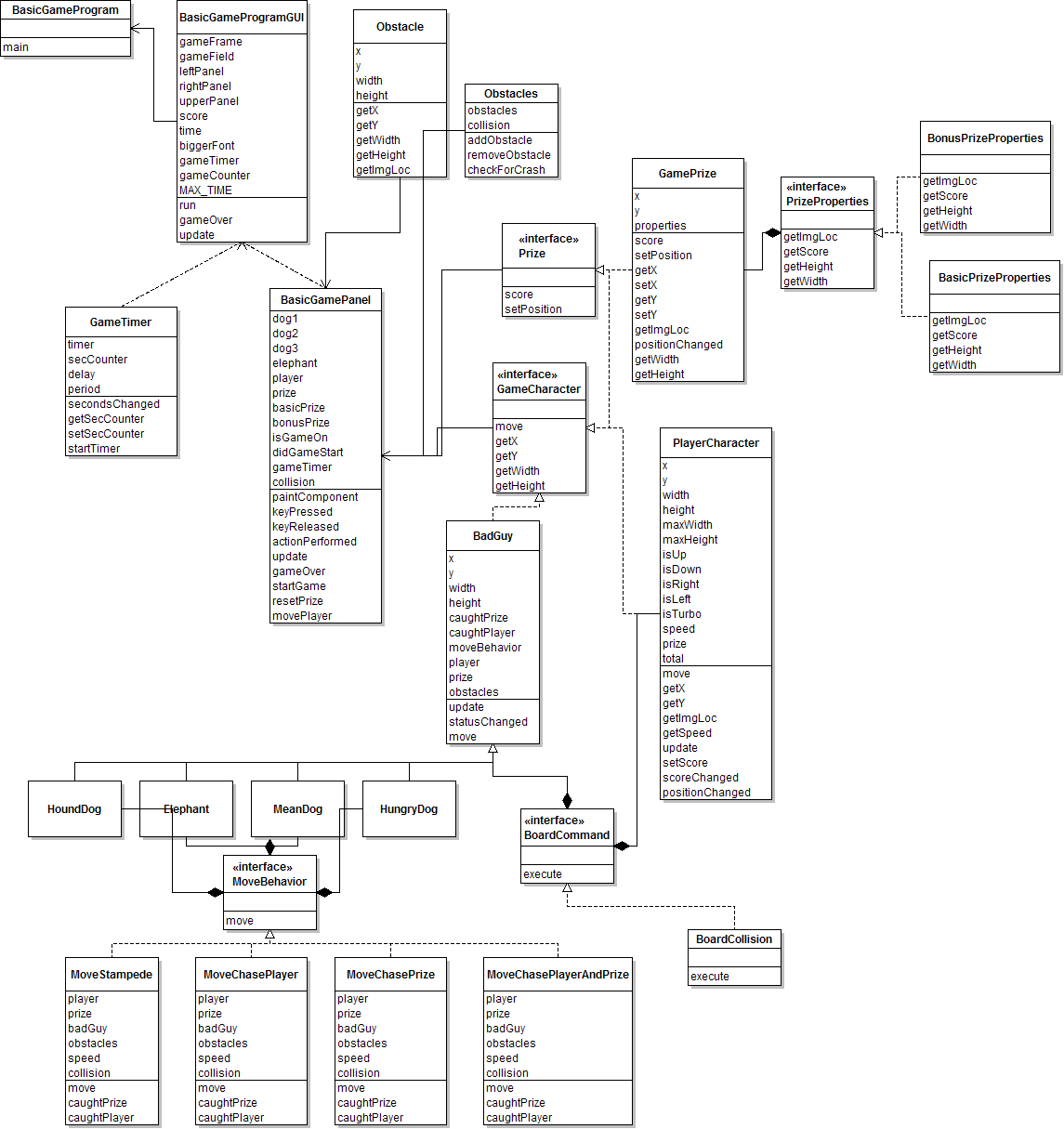
|  |
| --- |
| **Class List** |
| BadGuy |
| BasicGamePanel |
| BasicGameProgram |
| BasicGameProgramGUI |
| BasicPrizeProperties |
| BoardCollision |
| BoardCommand |
| BonusPrizeProperties |
| Elephant |
| GameCharacter |
| GamePrize |
| GameTimer |
| HoundDog |
| HungryDog |
| MeanDog |
| MoveBehavior |
| MoveChasePlayer |
| MoveChasePlayerAndPrize |
| MoveChasePrize |
| MoveStampede |
| Obstacle |
| Obstacles |
| PlayerCharacter |
| Prize |
| PrizeProperties |



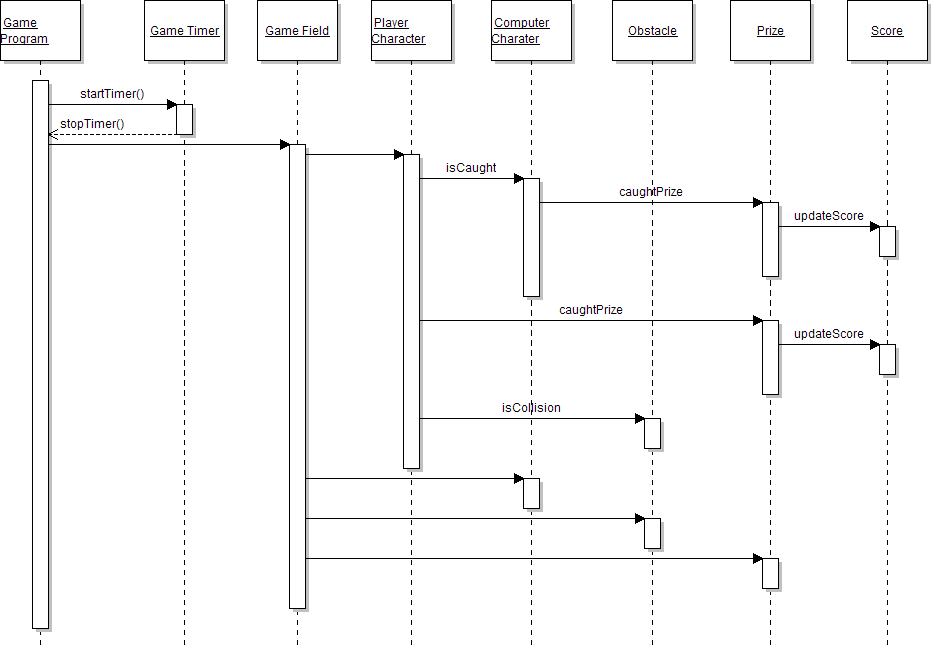
**Class Design**

|  |  |
| --- | --- |
| **Class Descriptions** |  |
| BadGuy | Abstract class, implements GameCharacter interface, provides the outline and implementation for the computer controlled characters. BadGuy is a subject in the Observer pattern. |
| BasicGamePanel | JPanel that builds the game field and is the controller for the main game action. An observer of both PlayerCharacter and BadGuy |
| BasicGameProgram | Launches the application |
| BasicGameProgramGUI | Called by BasicGameProgram. Starts the game timer thread and the game play thread. |
| BasicPrizeProperties | Implements the PrizeProperties interface. Is a decorator to GamePrize objects. |
| BoardCollision | Implements the BoardCommand interface. Provides a command that classes can execute to determine when objects collide on the play field. |
| BoardCommand | Interface that provides the outline for BoardCommand objects. |
| BonusPrizeProperties | Implements the PrizeProperties interface. Is a decorator to GamePrize objects. |
| Elephant | Extends the BadGuy abstract class. Elephant is one of the types of computer controlled characters. |
| GameCharacter | Interface that provides a basic common outline for all objects on the game field. |
| GamePrize | Implements both the Prize and GameCharacter interfaces. GamePrize is the object to be caught in game play. GamePrize is a subject in the Observer pattern. GamePrize is a lso a component in the Decorator pattern and provides an example of delegation. |
| GameTimer | GameTimer controls the game clock. It is a subject in the Observer pattern. |
| HoundDog | Extends the BadGuy abstract class. HoundDog is one of the types of computer controlled characters. |
| HungryDog | Extends the BadGuy abstract class. HungryDog is one of the types of computer controlled characters. |
| MeanDog | Extends the BadGuy abstract class. MeanDog is one of the types of computer controlled characters. |
| MoveBehavior | Interface that provides the outline for the different types of movements for computer controlled characters. MoveBehavior are implemented using the Strategy pattern. |
| MoveChasePlayer | Implements the MoveBehavior interface. Implements a type of movement where the computer character chases the player's character. |
| MoveChasePlayerAndPrize | Implements the MoveBehavioir interface. Implements a type of movement where the computer character chases either the prize or the player's character, depending on which is closer. |
| MoveChasePrize | Implements the MoveBehavior interface. Implements a type of movement where the computer character chases the prize. |
| MoveStampede | Implements the MoveBehavior interface. Implements a type of movement where the computer character randomly shoots across the play field. |
| Obstacle | Implements the GameCharacter interface. An Obstacle is a game character that does not move and provides an obstruction to other game characters. |
| Obstacles | Obstacles is a collection of Obstacle objects. |
| PlayerCharacter | Implements the GameCharacter interface. PlayerCharacter is the object that is controlled by the user. PlayerCharacter is both a subject and observer in the Observer pattern. |
| Prize | Interface that provides the outline for GamePrize objects. |
| PrizeProperties | Interface that provides the outline for PrizeProperty objects. |

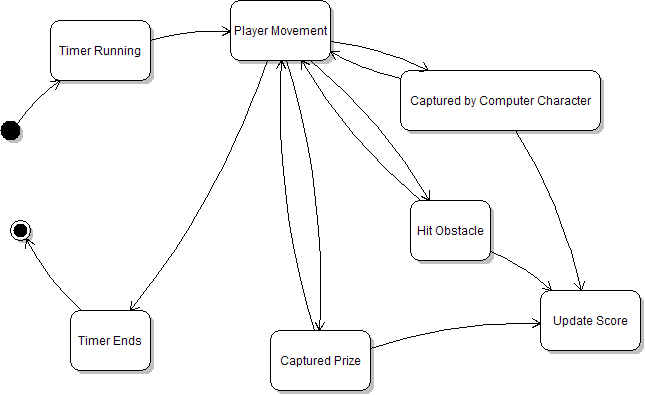
**Class Diagram:**



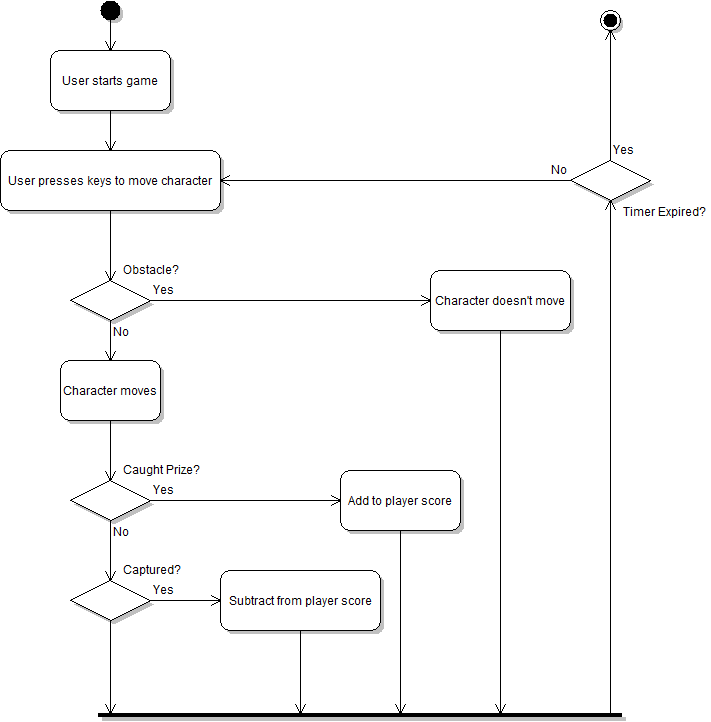
**Sequence Diagram:**



**State Diagram:**

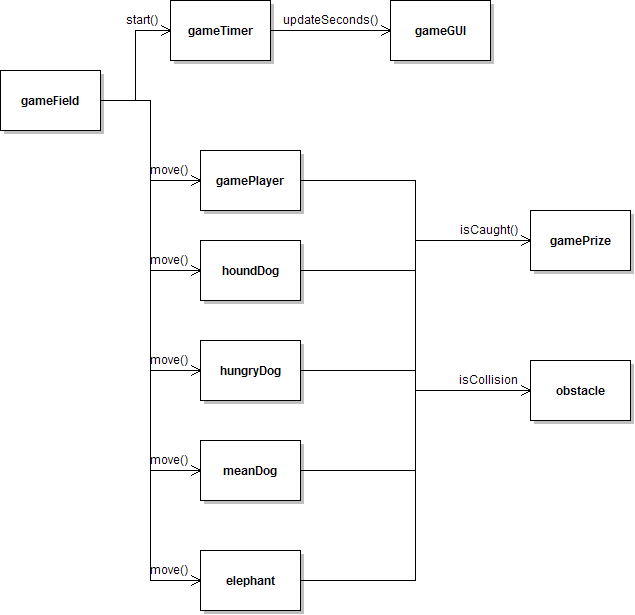


**Activity Diagram:**



**Collaboration Diagram:**

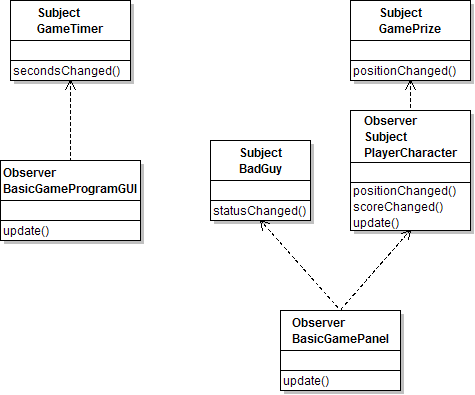
**Design Patterns**



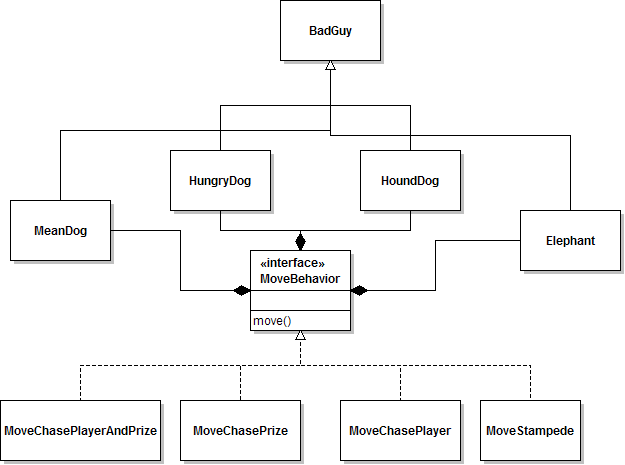
There are four design patterns that were applied during the development of this program. The Observer pattern is used so objects on the play field can notify each other when positions have changed and when one object has caught another. The Observer pattern is also used so the GameTimer can notify the GUI of changes in time. The Strategy pattern was applied to provide different types of movement behaviors for the different implementation of the BadGuy class. This makes it easy to change how a computer character will move or easily add a new computer character and assign a specific movement behavior. The Command pattern was applied to provide a specific command that all game characters can execute to check if it has collided with another game character. Finally, the Decorator was used with the GamePrize class so that GamePrize objects can be decorated with different GameProperties classes to easily allow for changes at run time that will create different prize types with different images and point values. The Decorator pattern makes it easy to add new prize types with new prize properties.

The decision to use the Command pattern was based on its ability to provide a broadcast mechanism that easily allows objects to communicate changes in status to other objects. This was an important feature necessary to the type of game play being used in this program. All other design patterns that were used were done so to provide a game program that is flexible and open to change.

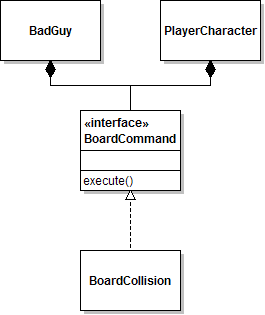
**Observer Pattern:**



**Strategy Pattern:**

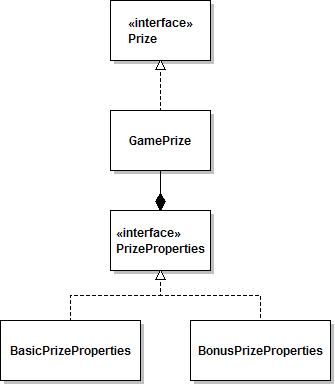


**Command Pattern:**



**Decorator Pattern:**

**Summary**



After the completion of this project I noticed that there were still a lot of areas in the program where proper design had not been applied. The main issue that I have with the final result is that Dependency Inversion is sorely missing. The fact that the program was initially built before I learned a lot of the proper design patterns along with the fact that I was a beginner at programming using Swing at the start of this project, I think contributed to this problem. I can see now a number of places in my design where there are unnecessary dependencies that can easily be removed. But I simply ran out of time to try to redo the code needed to make those changes.

At the beginning of the semester I thought that Model View Controller would be a logical design to use for a program with a GUI front end. I spent a lot of time researching MVC with Swing and came to the conclusion that it would be too difficult to implement with this project. I still think it is a pattern that makes sense with this type of application but I need more time researching MVC along with a better understanding of Swing.

Another area that I struggled with was design patterns that use abstract classes and sub classes that extend those abstract classes. The problem I ran into was applying that to the Swing classes that were already extending Swing components. For example, I applied the Observer pattern to this project with multiple subjects and observers. It seemed like a situation that needed the Mediator pattern to more efficiently manage the multiple subjects. However, all of the examples of Mediator pattern that I found involved classes that extend the Java Threads class. So the issue of multiple inheritance in Java I found made it difficult for some design patterns to work together. I’m sure there is a way to make it work using interfaces instead of extending classes, but I was unable to figure it out for this project.

In the end I thought that this project was a great learning experience. I learned as much from what I did not get to work as from what I did get to work. With what I learned through the course of this class I would definitely approach this project differently from the beginning. However I am happy with the end result of what I was able to accomplish. I was able to do more with Swing than I thought I would at the beginning of the semester. And more importantly, I did successfully implement four different design patterns that did help to turn the final result into a flexible and well designed program.

**References**

Icons and images used in this program were obtained from the following sources:



*Downloaded from:* [*http://www.iconspedia.com*](http://www.iconspedia.com)



*Downloaded from:* [*http://www.clker.com*](http://www.clker.com)



*Downloaded from:* [*http://www.iconspedia.com*](http://www.iconspedia.com)



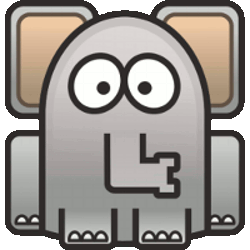
*Downloaded from:* [*http://www.iconarchive.com*](http://www.iconarchive.com)



*Downloaded from:* [*http://www.freeiconsweb.com*](http://www.freeiconsweb.com)



*Downloaded from:* [*http://www.freeiconsweb.com*](http://www.freeiconsweb.com)



*Downloaded from:* [*http://www.iconspedia.com*](http://www.iconspedia.com)



*Downloaded from:* [*http://www.iconspedia.com*](http://www.iconspedia.com)