Final Group Project: Arcade-style Pong

Authors:
Ben Whittaker (100481579)
Rob Edmonds(100489754)
Alec Crawford (100569102)

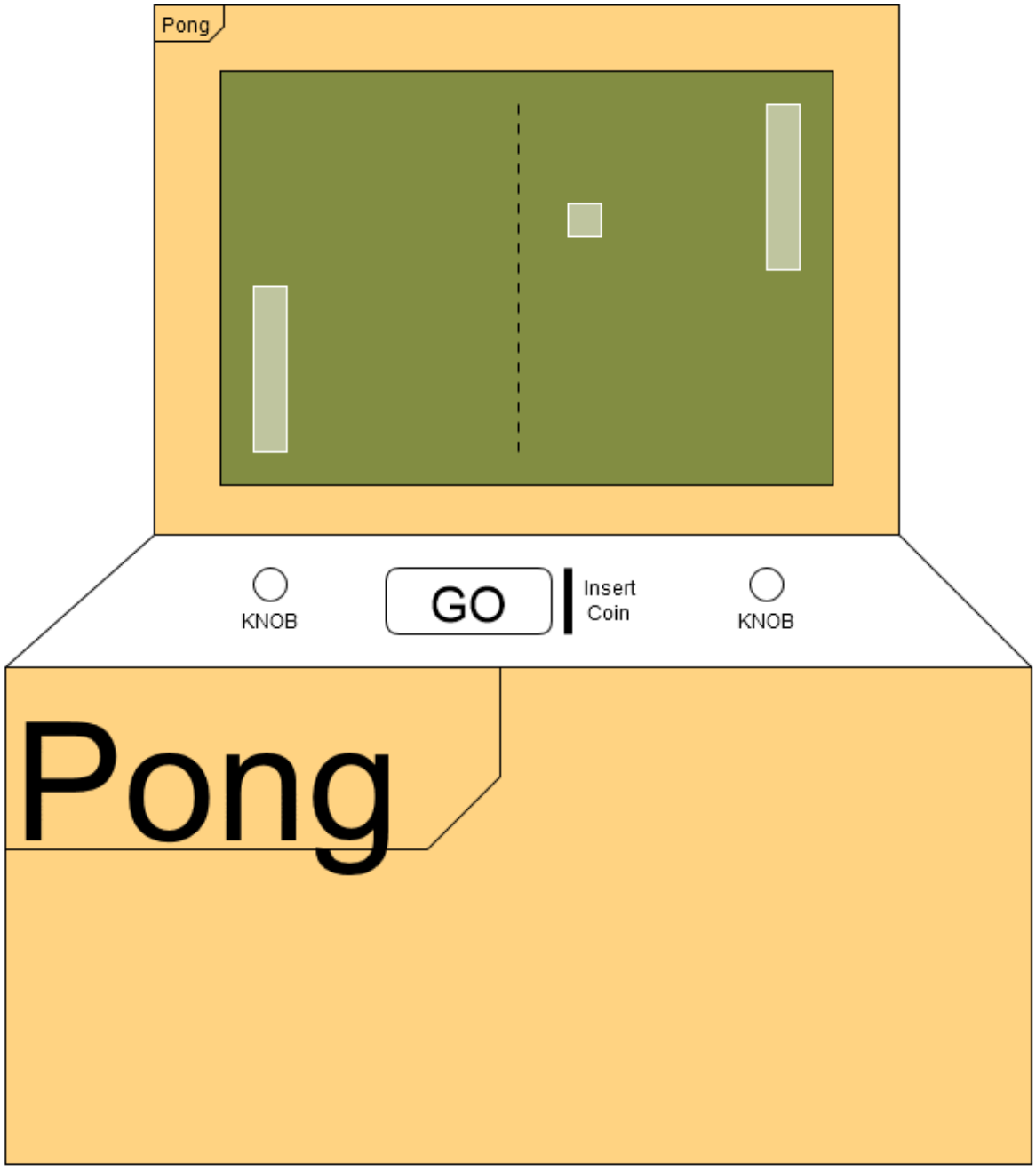
Course:
CSCI 2040U

Thursday, April 9th, 2015

Table Of Contents

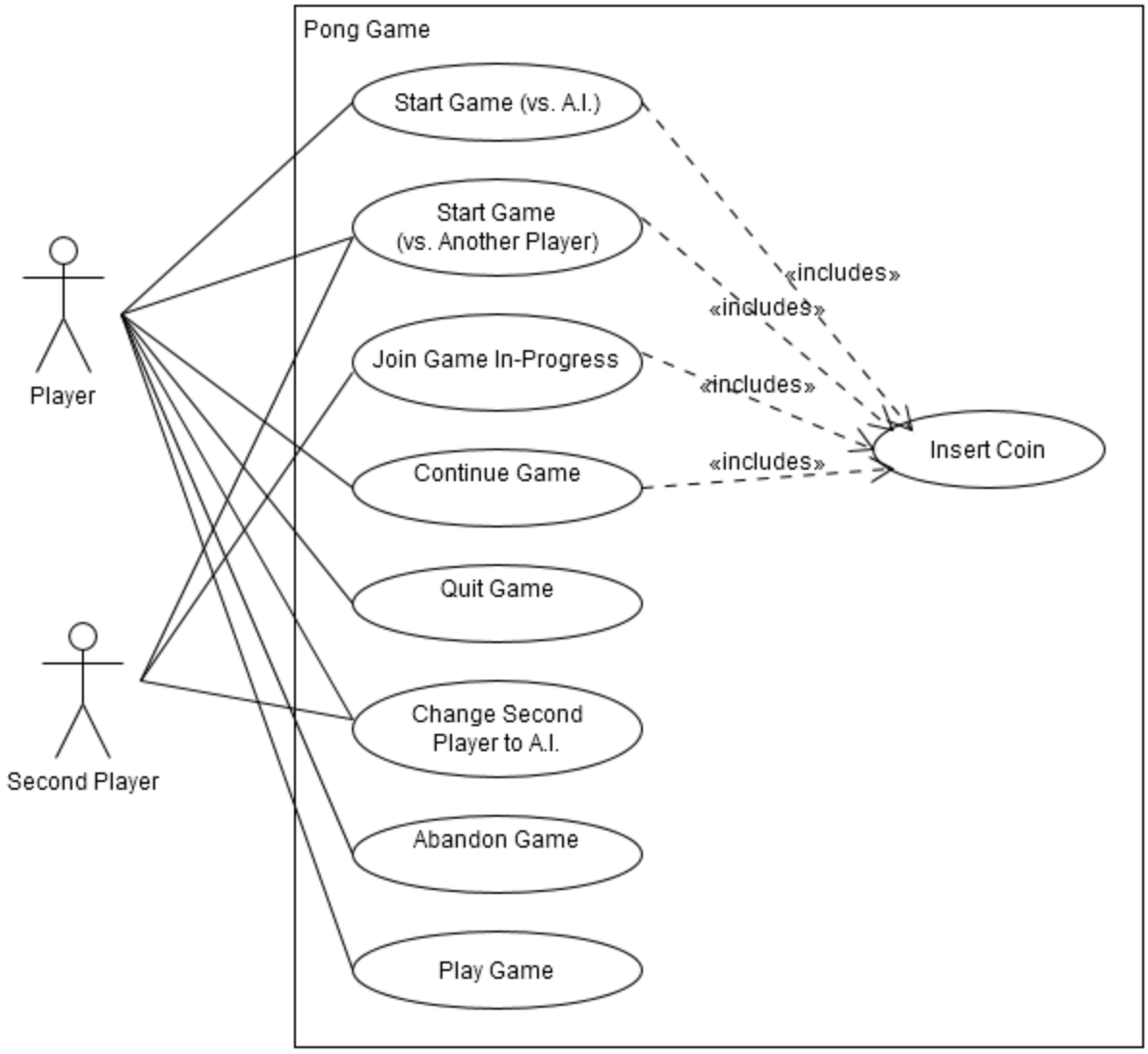
Topic

Our topic is pong, as it would be developed for an arcade machine. Below is the arcade machine our project would run on, mocked up in Umletino.



Use Case Model

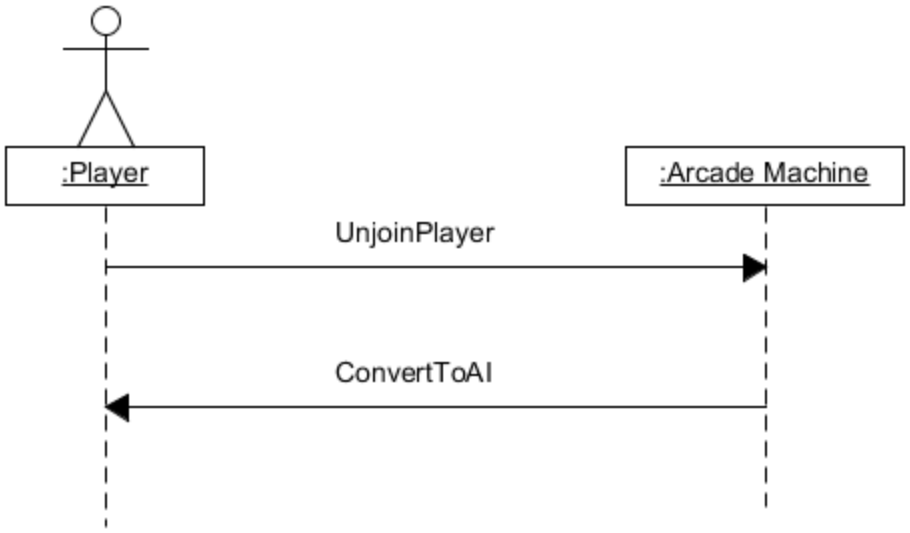
Use Case Diagram



Use Cases

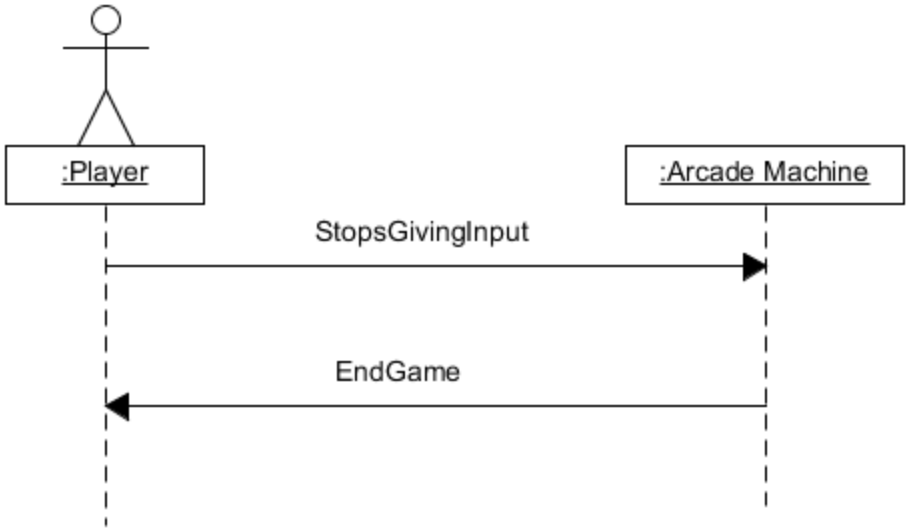
Use Case:	Change Second Player Avatar to an A.I.
Primary Actor:	First Player
Secondary Actors:	Second Player
Stakeholders and Interests:	
Frequency of Occurrence:	
Occasionally, when a second player has been playing the game and leaves without the first player.	

Main Success Scenario:	
	1. Second Player leaves the arcade machine. **2**. First Player press the second player's join/unjoin button.
3. The second avatar stops responding to the second knob input to the machine and starts to move based on A.I. behavior.	
	4. First Player continues to play the game against the game's A.I.
Alternative Flows:	
Technical Issues:	



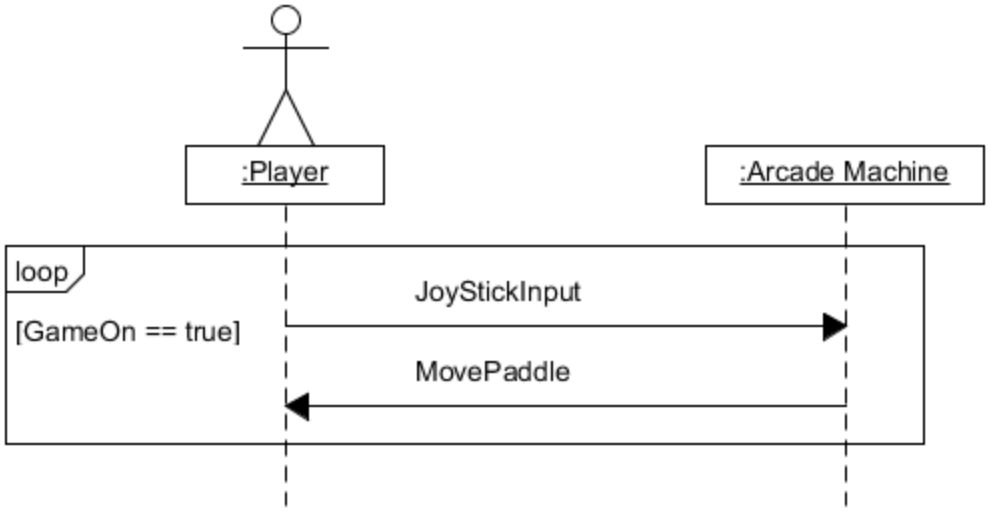
Use Case:	Abandon Game
Primary Actor:	First Player
Secondary Actors:	None
Stakeholders and Interests:	None
Frequency of Occurrence:	
Occasionally, when the player leaves the arcade machine before the game is over.	

Main Success Scenario:	
2(*). The arcade machine starts a timer to check long it has been since the machine has received any input. **3**. After a specified period of time, the game ends and returns to state where players would have to insert coins to play the game.	**1**. First Player leaves the arcade machine.
Alternative Flows:	
2a. Input is received from either one of the joysticks or one of the buttons to indicate the game was not abandoned.	
Technical Issues:	



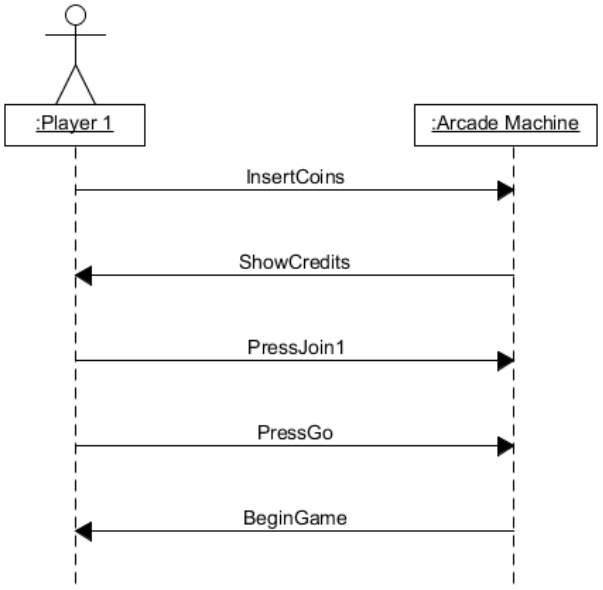
Use Case:	Play Game
Primary Actor:	Player
Secondary Actors:	
Stakeholders and Interests:	

Other Player – If there is a second player, how they choose to move their paddle depends on how the first player moves their paddle. **Frequency of Occurrence:** Occasionally, when a second player has been playing the game and leaves without the first player.	
Main Success Scenario:	
2. The arcade machine takes the spin direction of the knob as input and changes the velocity of the paddle depending on the direction the knob spins.	**1**. Player uses knob to control the direction their paddle moves. 3. The player sees their paddle at its new position
Alternative Flows:	



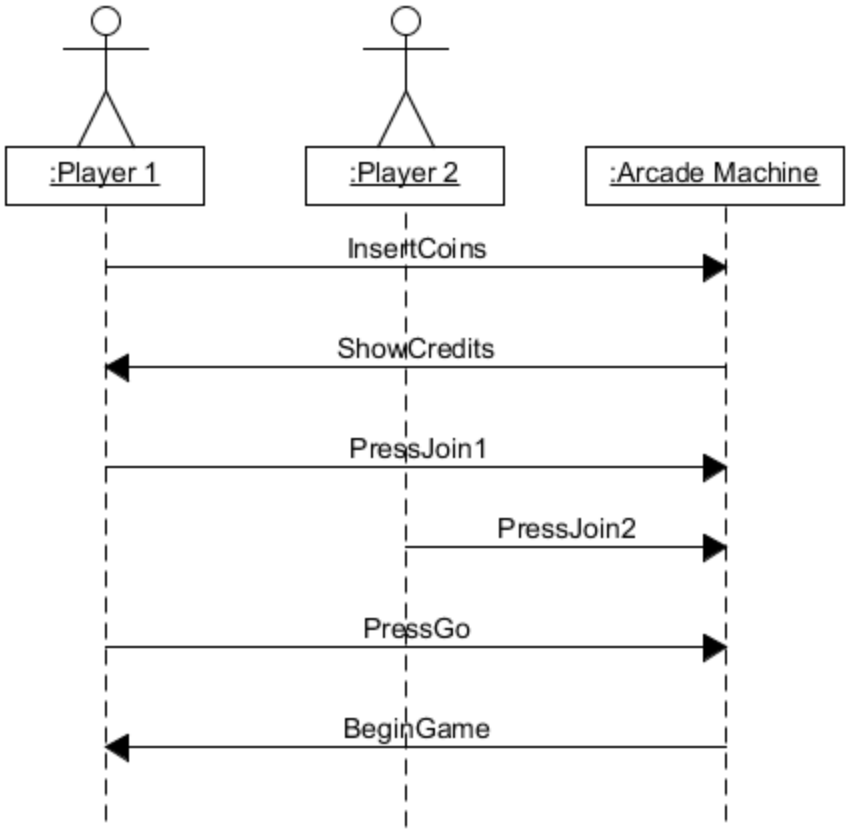
Use Case:	Start Game (vs. AI)
Primary Actor:	Player
Stakeholders and Interests:	
Frequency of Occurrence: Often (about half the games started)	

Main Success Scenario:	
2. The screen displays the number of coins inserted as ‘credits’ 4. The ‘Go’ button lights up to indicate the game is ready to be played. 6. The game begins.	1. Player inserts coins into the slot. 3. Player presses the ‘Join’ button on his side of the screen. 5. Player presses the ‘Go’ button.
Alternative Flows:	
1a. Player does not insert enough coins. The game cannot continue. 3b. Player continues inserting additional coins, looping back to step 1.	
Technical Issues:	

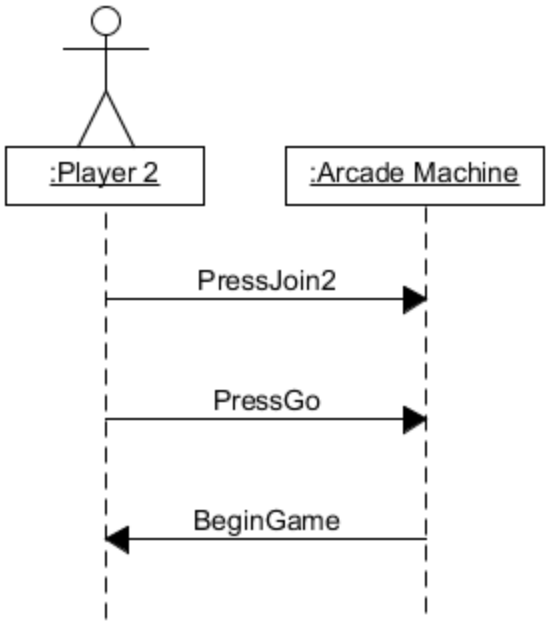


Use Case:	Start Game (vs. another Player)
Primary Actor:	Player 1
Secondary Actors:	Player 2
Frequency of Occurrence:	

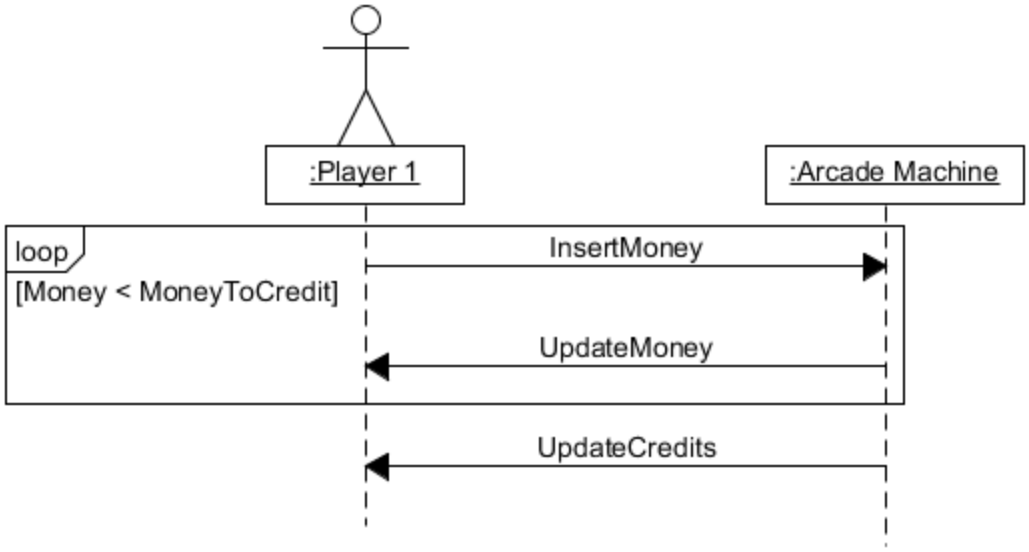
Often (about half the games started)	
Main Success Scenario:	
2. The screen displays the number of coins inserted as credits. 4. The ‘Go’ button lights up. 6. The screen changes to indicate 2 player mode. 8. The game begins.	1. Player 1 or 2 inserts a coin into the coin slot. 3. Player 1 presses the ‘Join’ button on his side of the screen. 5. Player 2 presses the other ‘Join’ button. 7. Player 1 presses the ‘Go’ button.
Alternative Flows:	
1a. Player 1 or 2 does not insert enough coins. The game cannot be played. 5c. Player 1 presses the 'Go' button. The game begins, but in vs. ai mode.	
Technical Issues:	


:Player 1
:Player 2
:Arcade Machine
InsertCoins
ShowCredits
PressJoin1
PressJoin2
PressGo
BeginGame

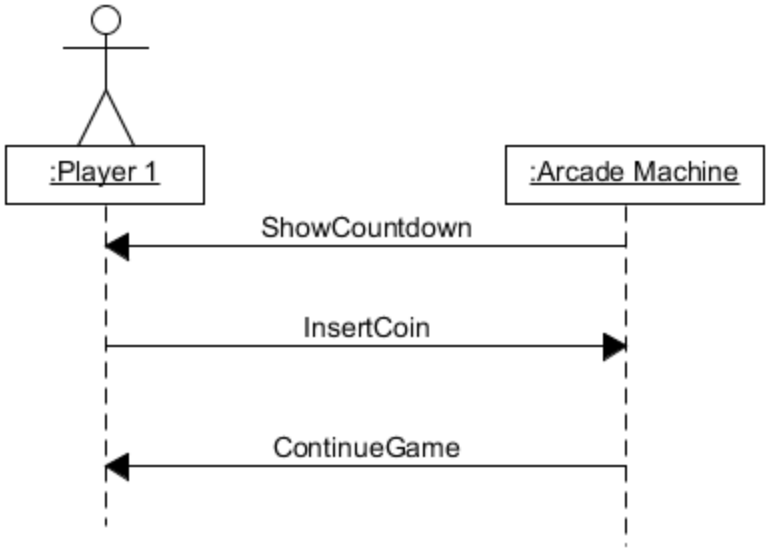
Use Case:	Join Game In-Progress
Primary Actor: **Secondary Actors:** **Stakeholders and Interests:** **Frequency of Occurrence:** Occasionally	Second Player First Player
Main Success Scenario:	
4. The screen changes, showing that the game is ready to be switched to 2-player mode. 6. The score is reset, and the game begins again in 2-player mode.	1. Player 1 starts a game vs. the AI. 3. Player 2 presses the Join button. 5. Player 2 presses the 'Go' button.
Alternative Flows:	
2a. Player 2 does not insert enough coins. Player 2 cannot join until sufficient money is produced.	
Technical Issues:	



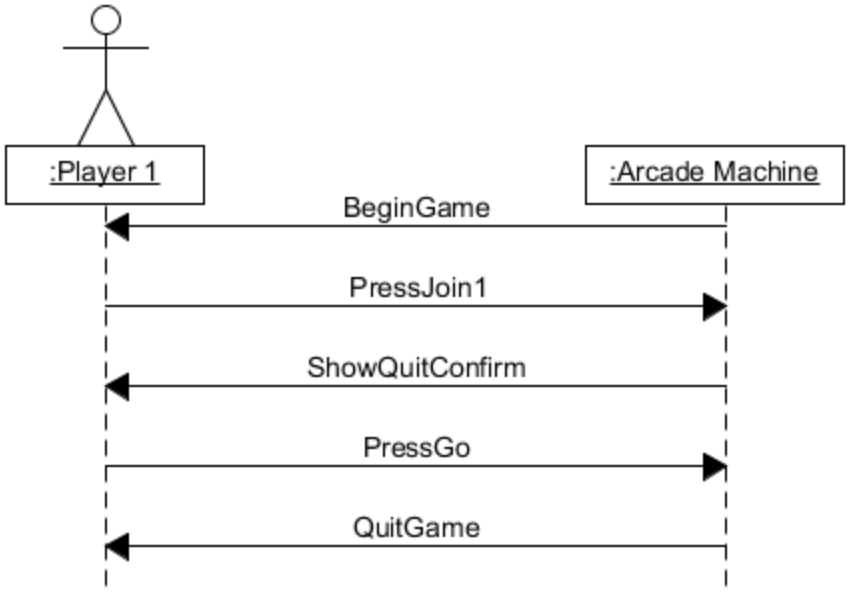
Use Case:	Insert Coin
Primary Actor: Player **Stakeholders and Interests:** **Frequency of Occurrence:** All the time	
Main Success Scenario:	
2. The machine displays the number of coins inserted. 3. When the number of coins exceeds the required cost, a 'credit' is added.	1. Player inserts coins into the machine
Alternative Flows:	
3a. Player does not insert enough coins to reach a credit. The machine remembers how many coins have been inserted.	



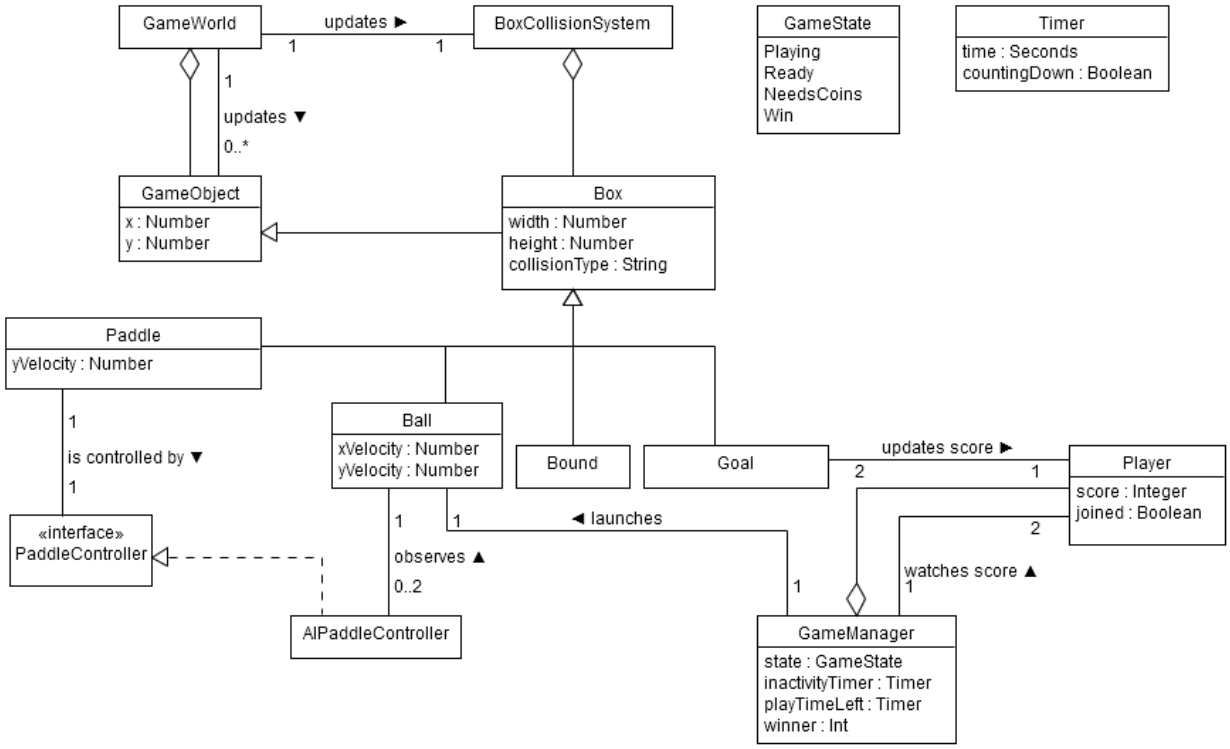
Use Case:	Continue Game
Primary Actor: Player **Stakeholders and Interests:** **Frequency of Occurrence:** Occasionally	
Main Success Scenario:	
1. The player's credits expire, rendering the game paused. A countdown begins. 3. The countdown ends, and the game continues.	2. Player inserts coins into the machine.
Alternative Flows:	



Use Case:	Quit Game
Primary Actor: Player **Secondary Actors:** **Stakeholders and Interests:** **Frequency of Occurrence:** Occasionally	
Main Success Scenario:	
3. The screen displays a confirmation that the player wants to quit. 5. The game finishes.	1. Player is playing a single-player game. 2. Player presses their 'Join' button. 4. The player presses the 'Go' button.
Alternative Flows:	
4a. The player presses the 'Join' button again. The game, instead, resumes.	



Domain Model



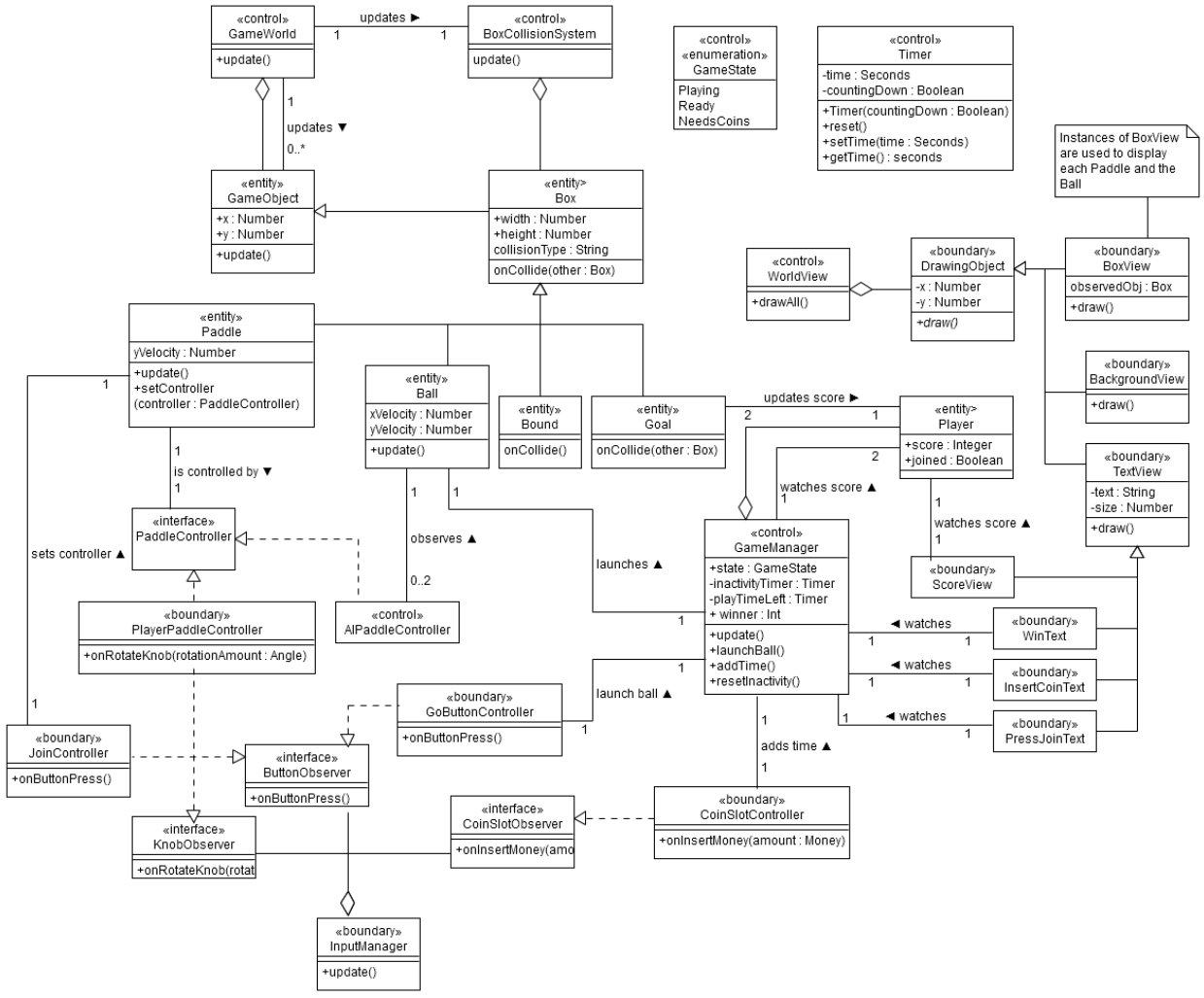
Architecture

The architecture used by this particular program is MVC. For an arcade game like this, code is needed to handle the movement of the game controls and the display of objects on the screen, in addition to the core code needed to run the game. Therefore, MVC is the best design architecture to use in this case.

In order to implement the MVC architecture, separate classes exist for the representations of objects in the code, how those objects are controlled, and how those objects are displayed. As an example, with the paddles that reflect the ball, control of the paddle is handled by the PaddleController interface, management of the variables of the paddle is handled by the Paddle class, and display is handled through the BoxView class. Clear separation of these features lend to moderate coupling, but easy readability and clarity of the system.

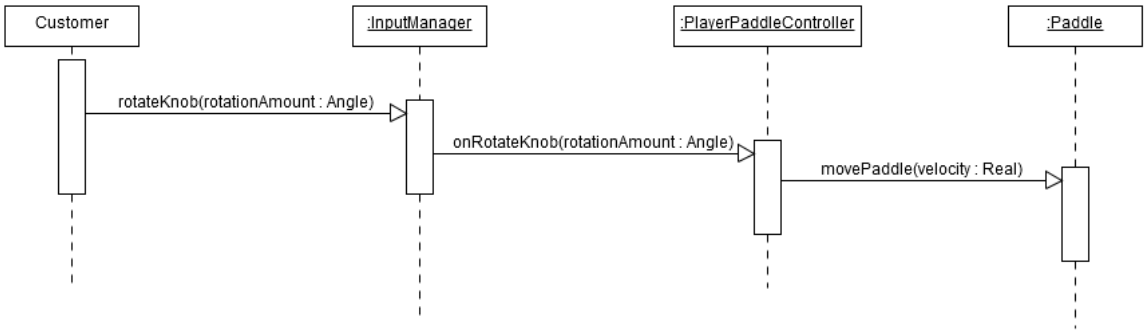
Design Model

Design Model Diagram

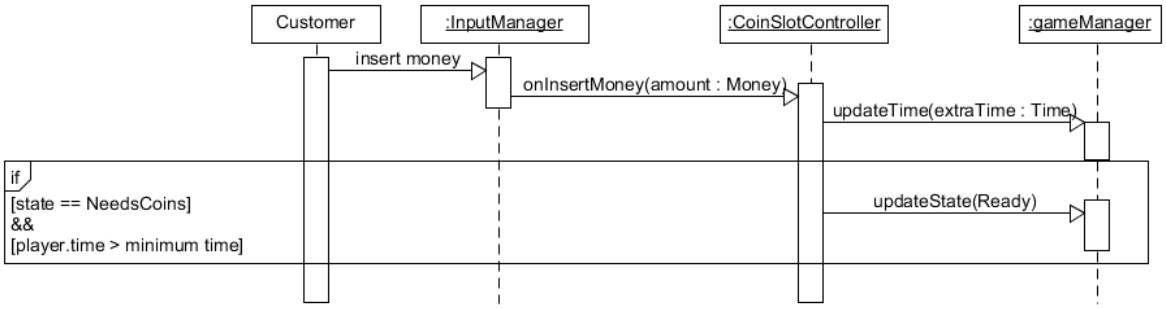


Sequence Diagrams

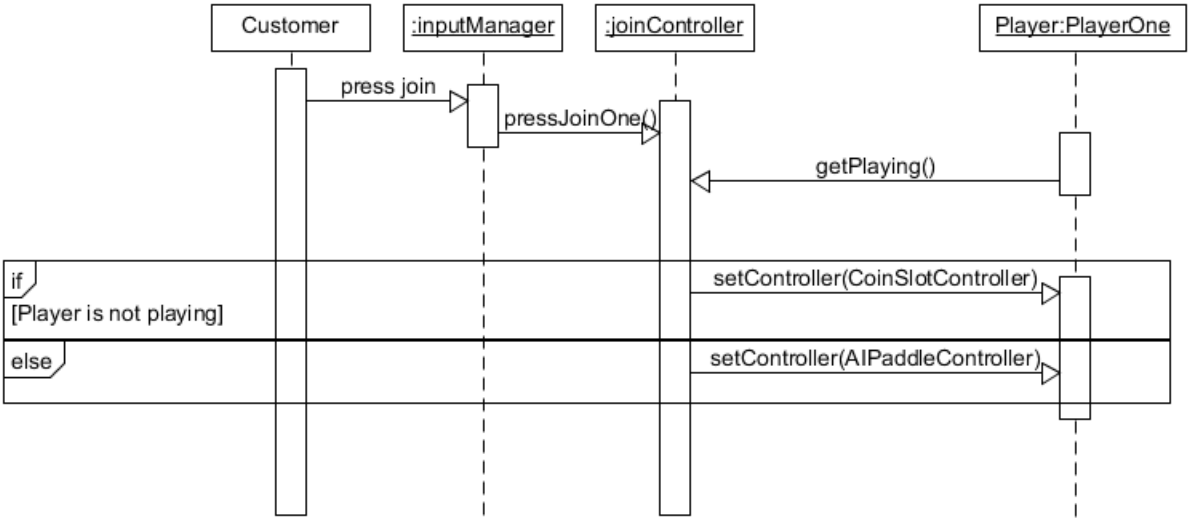
movePaddle()



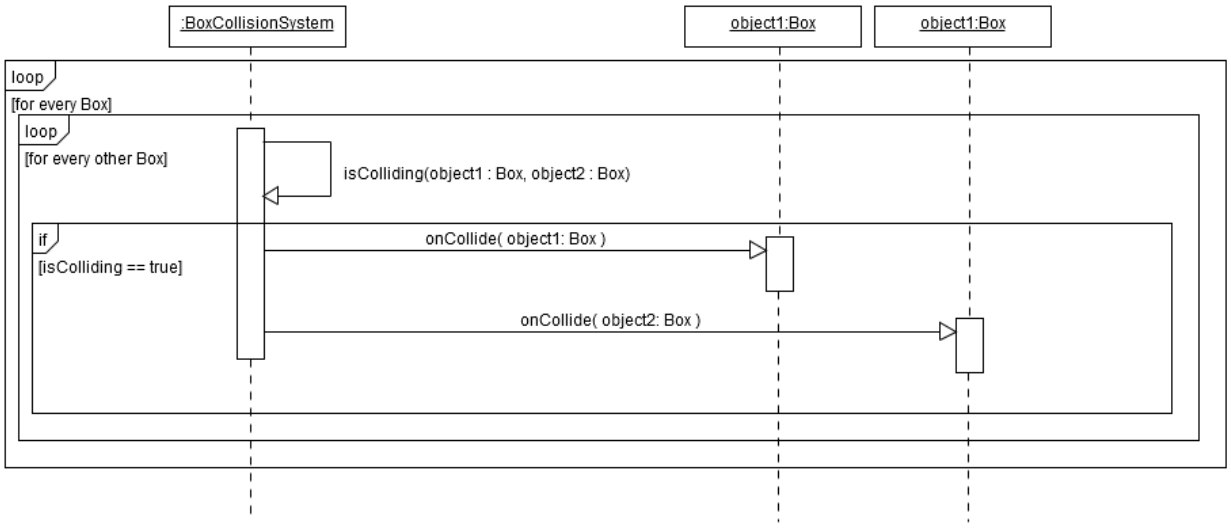
insertCoin()



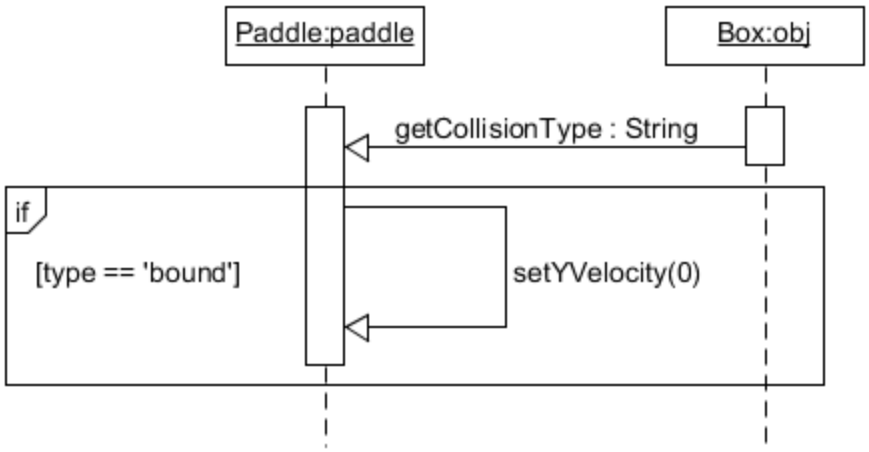
joinPressed()



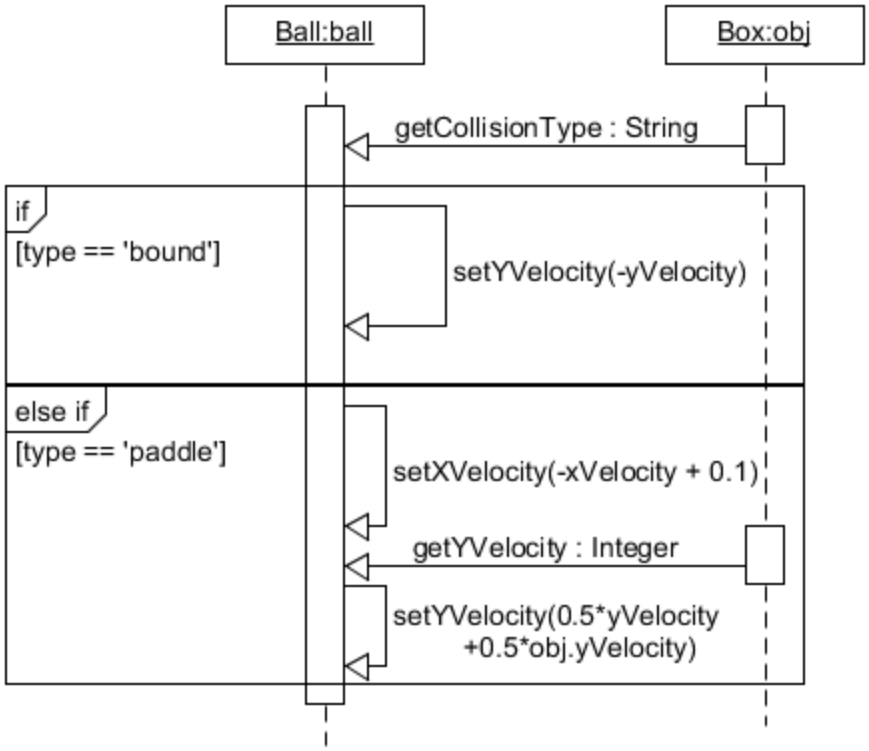
detectCollision(object1 : Box, object2 : Box)



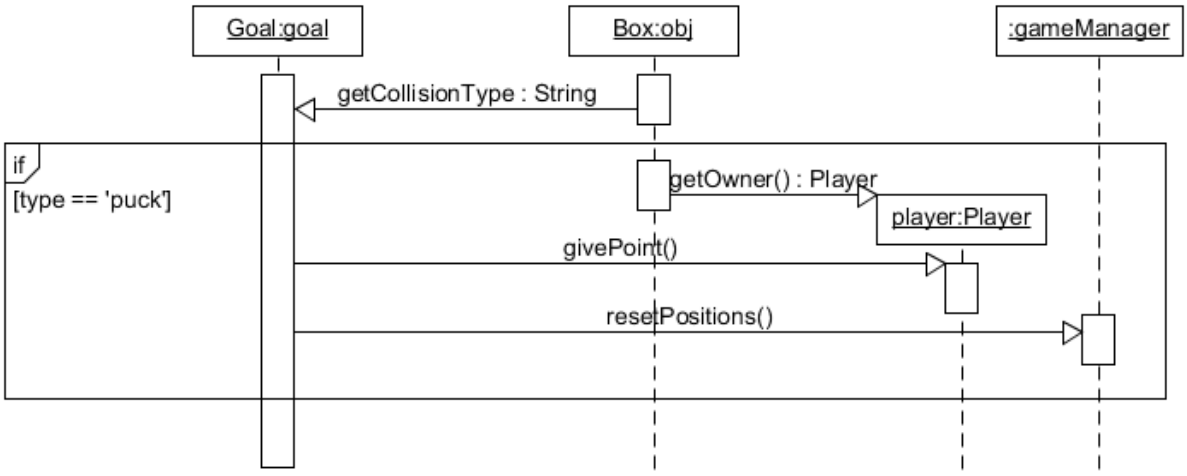
paddle.onCollide(obj : Box)



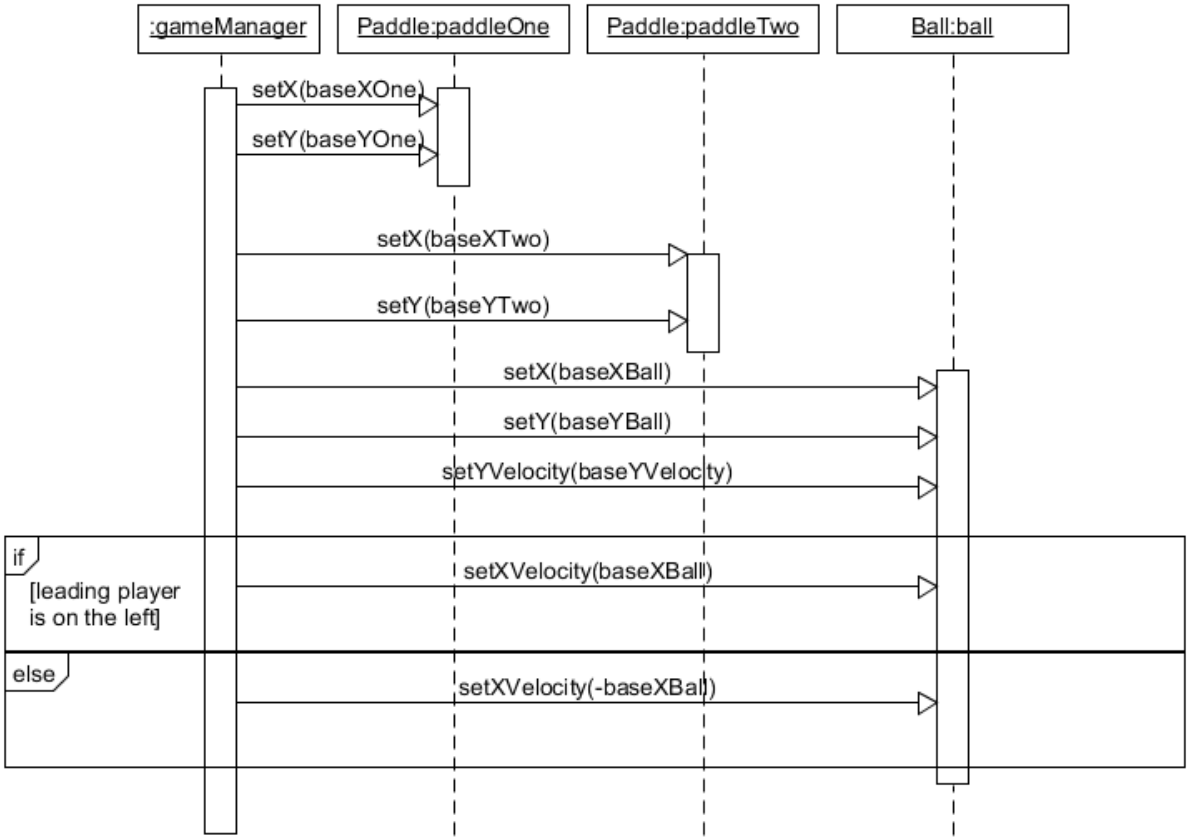
ball.onCollide(obj : Box)



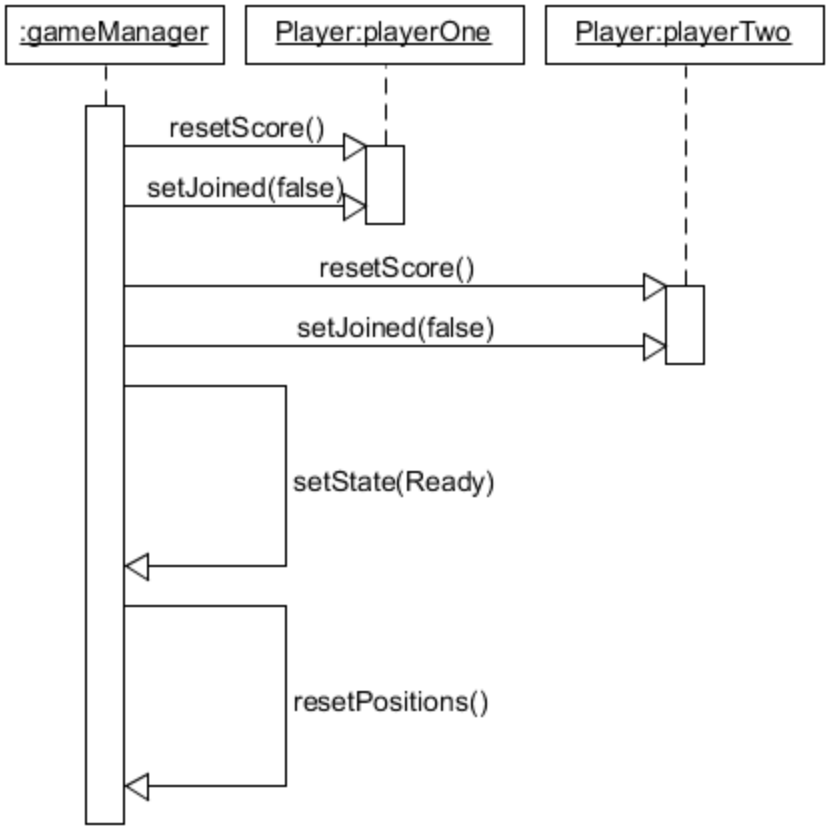
goal.onCollide(obj : Box)



resetPositions()



resetGame()


:gameManager
Player:playerOne
Player:playerTwo
resetScore()
setJoined(false)
resetScore()
setJoined(false)
setState(Ready)
resetPositions()

Implementation

Adapter Pattern (AI Paddle Controller)

Implemented by Alec Crawford

AIPaddleController.java

```
// Alexander Crawford
class AIPaddleController implements PaddleController {
	Paddle paddle;
	void movePaddle(int distance){
		// Code to calculate the movement goes here. It can be called by the ball.
	}
}
```

PaddleController.java

```
// Alexander Crawford
interface PaddleController {
	Paddle paddle; // the paddle
	void movePaddle(int distance); //the interface provides an alternative way to move the paddle
}
```

Paddle.java

```
class Paddle implements Box {
	int yVelocity;
	PaddleController controller;
	void update(){
		//Update the paddle's position.
		this.setY(yPaddle + yVelocity);
		yVelocity *= 0.75; // reduce velocity
	}
	void setController(PaddleController newController){
		controller = newController;
		newController.paddle = this;
	}
}
```

Observer Pattern (Button Input Observers)

Implemented by Ben Whittaker

ButtonObserver.java

```
// Ben Whittaker
```

```
interface ButtonObserver
{
	public void onButtonPress();
}
```

InputManager.java

```
// Ben Whittaker

import java.util.*;

// This class is currently only implemented to deal with button input, but in
// the real world it would handle the coin slot and the knob inputs as well.
public class InputManager
{
	InputManager()
	{
	joinOneObservers = new ArrayList<ButtonObserver>();
	joinTwoObservers = new ArrayList<ButtonObserver>();
	goButtonObservers = new ArrayList<ButtonObserver>();
	}

	public void update()
	{
		// This class would poll the hardware for the button states,
		// and call onButtonPress on the observers in the lists as appropriate.
		// I have not implemented this as our design does not extend to the
		// hardware interface.
	}

	public void registerJoinOneObserver(ButtonObserver observer)
	{
	joinOneObservers.add(observer);
	}
	public void registerJoinTwoObserver(ButtonObserver observer)
	{
	joinTwoObservers.add(observer);
	}
	public void registerGoButtonObserver(ButtonObserver observer)
	{
	goButtonObservers.add(observer);
	}
```

```
        private List<ButtonObserver> joinOneObservers;
        private List<ButtonObserver> joinTwoObservers;
        private List<ButtonObserver> goButtonObservers;
}
```

GoButtonController.java

```
// Ben Whittaker

public class GoButtonController implements ButtonController
{
        GoButtonController(GameManager manager)
        {
        this.manager = managaer;
        }

        public void onButtonPress()
        {
        manager.launchBall();
        }

        private GameManager manager;
}
```

Template Method Pattern (Drawing)

Implemented by Robert Edmonds

DrawingObject.java

```
// Robert Edmonds

public abstract class DrawingObject {
   public DrawingObject(float PosX, float PosY) {
         x = PosX;
         y = PosY;
   }

   public abstract void draw();

   private float x;
   private float y;
}
```

BackgroundView.java

```
// Robert Edmonds
```

```
public class BackgroundView extends DrawingObject {

    public BackgroundView(float PosX, float PosY) {
        super(PosX, PosY);
    }

    public void draw (){
        //draws the rectangles at the borders and around the goal area
        //relative to x and y
    }
}
```