Lex Steffens  
  
Multiplatform development design document  
  
General concept  
I chose to create a simple pong style game with two different input methods. The goal is for this game to be playable with keyboard button controls or Xbox 360 controller input. This game was developed in Monogame using C# scripts.  
  
Key points  
I created the MultiplatformInputHandler() class. This class is completely seperate from the rest of the game logic. If the control method needs to be changed or updated this is the only class that needs to be worked on. It serves as a Wrapper for the game input logic.   
  
The MultiplatformInputHandler() class is the only class that registers input from the console at all. I made sure to include generic names for the game logic actions it encompasses. These include things like paddle1up to move the player 1 paddle up and startgame1 to enter the first variant of game state.   
  
If this game were to be developed for even more platforms only the MultiplatformHandler() class would need to be edited for input mechanisms.  
  
In terms of game design the movement for both paddles is not analog. It is instead of digital movement speed and therefore was developed to work the same for the xbox analog sticks.  
  
Reflection  
  
When developing this game I worked on creating a proper skeleton for the game logic first. This enabled me to keep the input mechanism seperate while developing the basic game. The controls were the last thing I added to the game and are confined in a single class for the multiplatform convenience. The code ends up running cross platform, with no need to run any different when a Xbox Controller is connected and vice versa. While I wanted to do more on input methods with ideas like perhaps touch screen controls. The basic idea of wrapping multiplatform code seperate from the rest of the game logic I was able to conduct with succes.  
  
  
  
  
  
  
  
  
  
  
  
When reconsidering the true meaning of the course “Multiplatform development”, I started realize the main point is to save on compilation space and ease of reusing code on other systems. By seperating game logic from machine code (like reading inputs and graphical display), A game is more easily supported by multiple systems and compilation styles. Trying a few different compilations, I finally stuck with the original concept of making a seperate version of the pong game for when using a XBOX Controller.   
  
In order to succesfully create checks for pre compilation I had to make use of Symbols. These symbols are essentially booleans determining what kind of code is to be compiled and what isn’t. By simply changes these boolean values between different platform builts, a simple yet cohesive game logic and code can be kept. After some digging I managed to create my own set of Symbols in the Monogame engine I used. Ending up with the pre existing Symbols WINDOWS and LINUX, as well as the created CONTROLLER and VIEW symbols.   
  
For the main two builds of this pong game, all that needs to be altered between builds is the definition of the CONTROLLER Symbol. This definition is at the very first line of the Game1.cs script file. It is undefined by default (#undef CONTROLLER), rendering the keyboard controls version. For the other build, this line is ommitted (//#undef CONTROLLER). In turn forming the xbox controller version build.   
  
The code within the Multiplatformhandler() class is never fully compiled. It will only compile the part neccessary for controlling depending on the defined state of the CONTROLLER symbol. Any other control methods could just as easily be added with a new Symbol.   
  
When developing the Symbols and various builds. I tried using some other symbols as well like LINUX, however I was unable to determine some proper differences for such a version as I myself don’t have access to a Linux system of sorts. I also had the idea of VIEW determining a windowed screen build of the game, but ended up not using it due to not getting a proper idea for it.  
  
Reflecting upon the revised version I feel I know truly understand the core concept of the course, adjusting code to the importance of disconnected game logic and surface level specific machine code resulting in managable multiplatform development.