15-466 Computer Game Programming
Fall 2011
Project 3

For this assignment you will be writing the AI for a character in a fighting game. The game is a simplified version of Street Fighter where the characters cannot move away from each other (there aren't left/right moves). You will apply techniques you've learning in class in order to build a character that plays well. The game will be played over the network so you can play against your classmates.

I. The Rules of the Game

There is a state machine in Figure 1 showing the rules of how your character behaves. In each of these states there is a chain of frames, which the character goes through. Each of the frames is vulnerable to different kinds attacks. For instance some of the frames during ducking can only be hit by a low attack. After explaining the state machine, we'll go over how each state plays out.

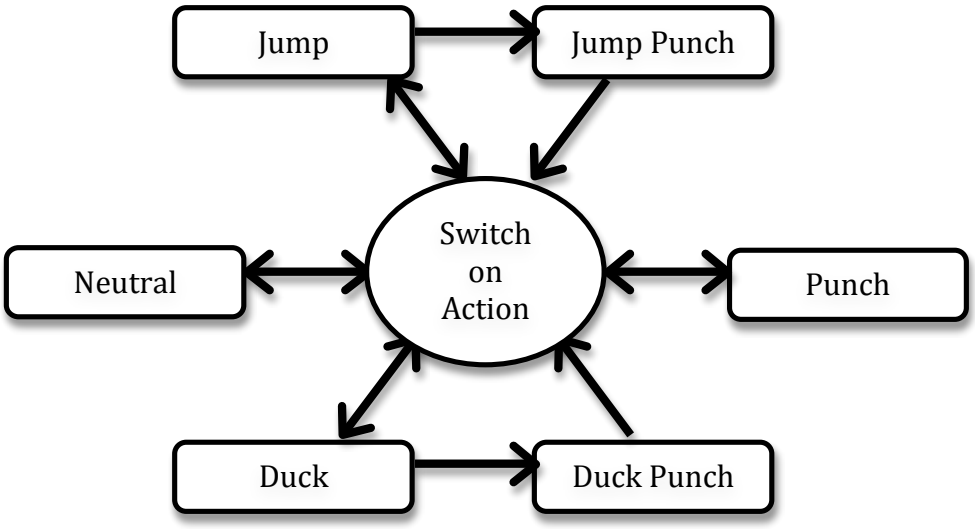


Figure 1: The state machine

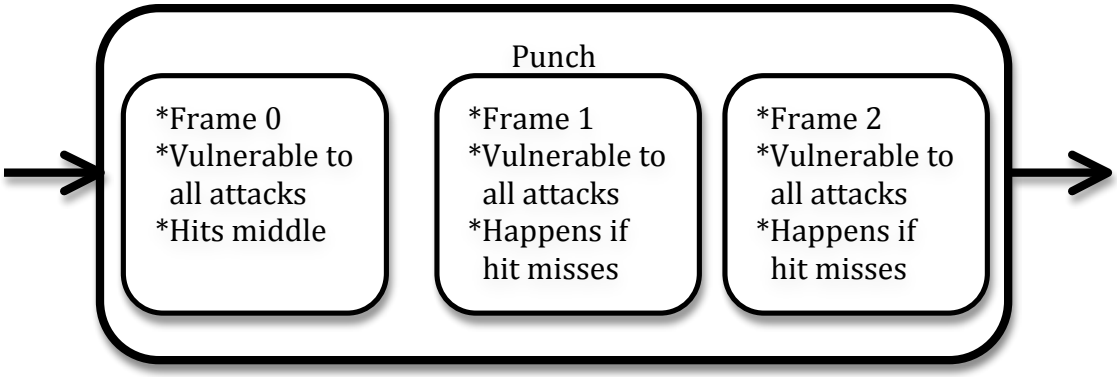
In Figure 1, each of the rectangular objects is a state with frames inside. The circular object in the middle is a switch indicating that if the state machine goes here it immediately goes to one of 4 states based on the action your player has most recently commanded. (This switch is not a state, I just didn't want to draw all the extra arrows.)

There are 4 actions you can perform in the game: jump, duck, punch, and none.
When using the "Switch on Action" the character follows these rules:
Action Punch takes you to state Punch
Action Jump takes you to state Jump
Action Duck takes you to state Duck
Finally you go to Neutral if you use action None or if you provide no action at all.

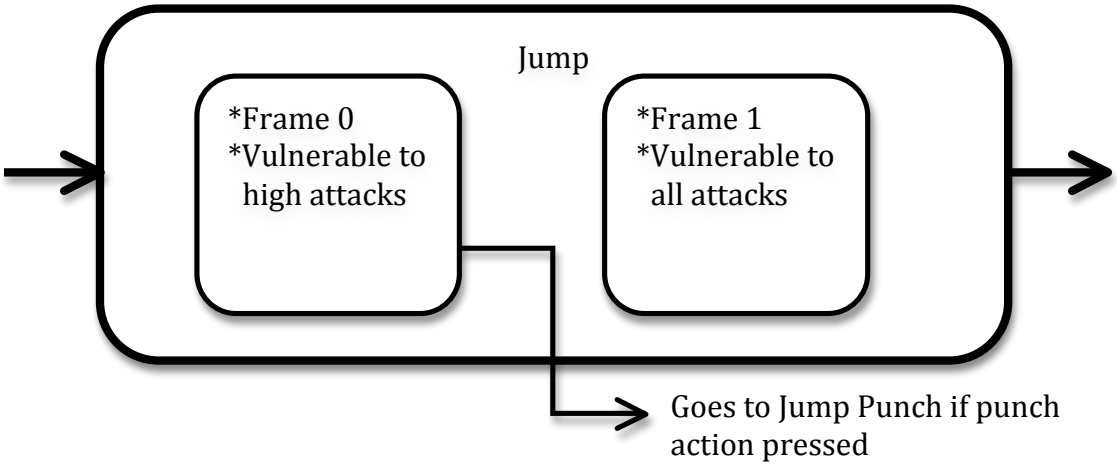
You can see that states “Duck Punch” and “Jump Punch” can only be reached by going to “Duck” or “Jump” respectively. In these states you must send the “Punch” action part way through the Duck/Jump in order to transition.

In the following figures, we explain what happens in each of the states.

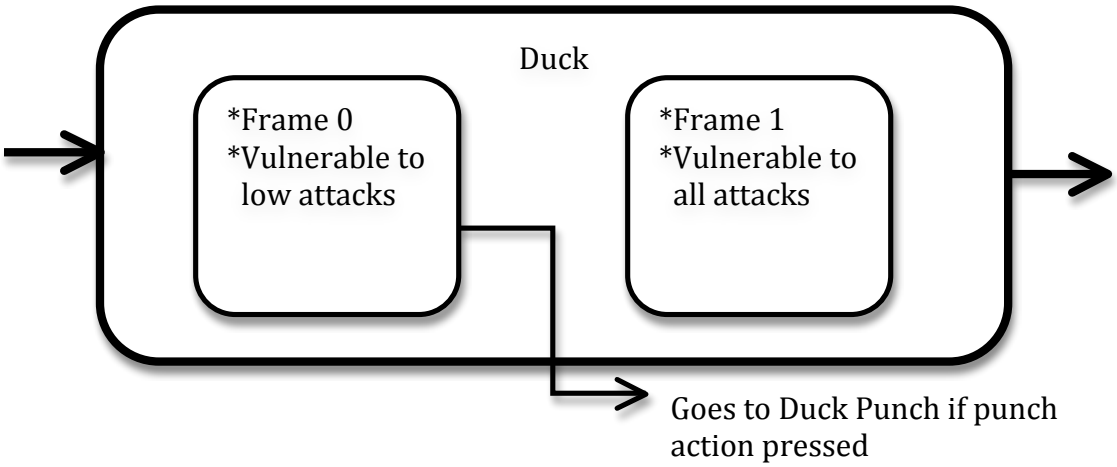
Punch – This performs a middle punch. On frame 0 it actually inflicts the damage. However, if the punch misses (by going over a ducked player or under a jumping player), the punch has **2 extra frames** (1-2) while your player is “off balance”.



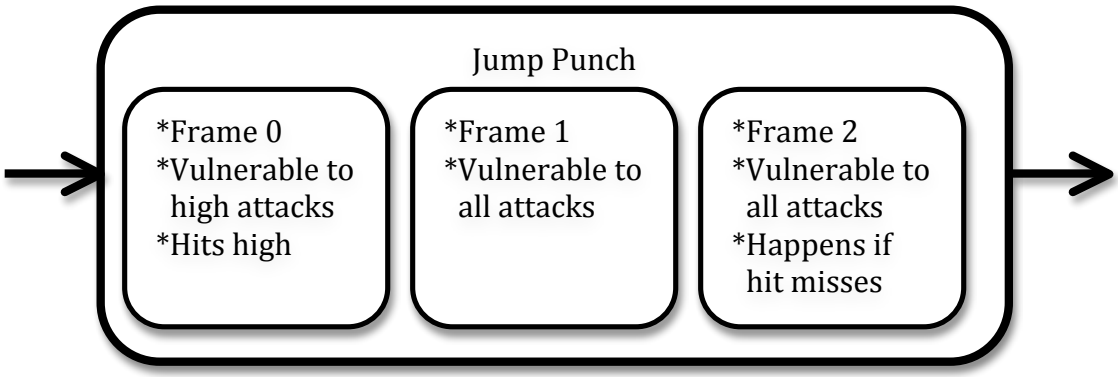
Jump – This state lasts the 2 frames and for the first one your character is in the air. When the character is in the air it can only be hit by a jumping punch (mid and duck punches miss). This state can be left after frame 0 if the punch action is pressed to transition into “Jump Punch”.



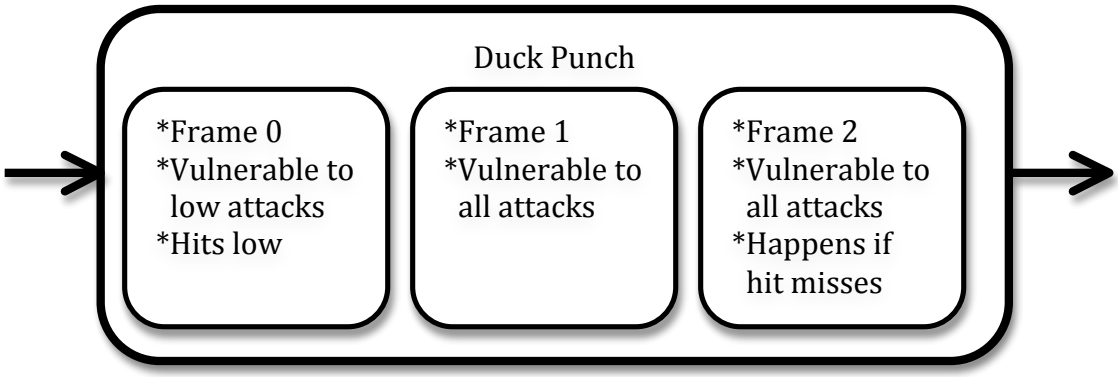
Duck – This state lasts the 2 frames and for the first one your character is ducking. When the character is ducked it can only be hit by a ducking punch (mid and high punches miss). This state can be left after frame 0 if the punch action is pressed to transition into "Duck Punch".



Jump Punch – The jump punch allows you to land a high hit, which can hurt a player who is in the air or standing at neutral. The hit happens on the first frame of the state and if the hit misses (the opponent is ducking) your player wastes an extra frame (2).



Duck Punch – The duck punch allows you to land a low hit, which can hurt a player who is ducking or standing at neutral. The hit happens on the first frame of the state and if the hit misses (the opponent is jumping) your player wastes an extra frame (2).



The game is played over the network and your code will be written in ClientScript.cs (though we will be giving you the server code as well). The server will be simulating the game and on a clock will execute each of the players' next frames in a synchronized manner. The server will receive an action from the 2 clients for the upcoming frame in the time period between the last frame and the upcoming frame. A frame will happen every 1s so this is the amount of time your code has to look at the game state from the previous frame and decide what it wants to do. Note: Sometimes there will be frames where you can't execute a new action because you have to finish the action you are currently doing (the server will ignore actions you send in this case).

Each player starts with 50 health and you lose 1 health each time you are hit. You win if you reduce your opponent's health to 0 or if you have the most health when the game timer runs out.

II. Playing the game

In order to build a winning player you can use any of the techniques discussed in class or anything else you think may be helpful. Learning to predict what your opponent will do may be helpful and for that you may want to use N-grams, naïve bayes, or reinforcement learning. Additionally, you could choose to build a decision tree or state machine.

In the ClientScript.cs you have to implement the function getPlayerAction, which returns the action you would like to send to the server. There are several variables you can use to make your decision, such as the action you and your opponent sent

last frame, the state (and frame in that state) that you and your opponent we on in the last frame, and the health for each of you.

You function is called to get your action and then it sends your result to the server using RPC. You should only need to send one action for every time you get the state. The server only uses your most recent action for the upcoming frame.

At the top the ClientScript.cs are three important variables, server_ip, server_port, and useKeyboardInput. The first two we will send out in an email for a server we will set up for you to play against your classmates (and possibly some bots we will make over the next few days). You should feel free though if you have access to several computers, to set up your own server (we provide the server code).
The last variable turns on or off keyboard input. If you set this to false, it will play using your code instead of the keyboard. If you want to play with the keyboard, the default keys are:
Up Arrow: Jump
Down Arrow: Duck
Z: Punch

III. Adding Chat
The networking is done with RPC (remote procedure call). This allows a computer to call a function on another computer. You can see examples of this in the transfer of the game state that we have implemented. Using RPC you will be able to call a function remotely on all of the other machines in the game (your opponent, and the server). You will implement a chat feature in your game so that when pressing "Enter" on the keyboard you can type a message and by hitting "Enter" again the message is sent to your opponent. You will also need to display the conversation between you and your opponent on the screen.

The signature for the chat function you will call over RPC is:
void chat(string s);

IV. Grading
60% - There will a set of bots we will run your player against. The number of points you get is based on how many games you win. We may release a subset of these bots to be played against while you are working on the assignment. The bots will have varying amounts of predictability, so that there will be something for your player to learn (if you use a learning technique).
10% - Implementing chat and displaying the conversation in your game.
30% - You will submit a **short paper** (no more than 1-2 pages) describing your approach and the things you tried out. You will also need to submit a **video** showing your player in action. Finally, you will get credit for how **general/robust** your approach is.
Bonus: We will run your player against all your classmates and you can get up to 10% extra credit based on your ranking within the class.