# ICS4U Network Design Document

Kaden S, Martin S

- All programs have a universal SSM Listener in GameState, which is put in each of the individual constructors
- Listener splits messages by their commas to get their specific actions into an array, where each index will then be assigned to the respective action

## Home View

- Based on the button that the user presses, the program will differentiate them as either the host or the client. Then, SSM instances are initialized based on this using ssm.connect() with the randomly picked port number of 4019

## Character Selection View

- SSM sends socketed text to the other player based on the character picked, which tells the game/program what characters to draw on the screen during the game and the name of the player
- The SSM Listener then listens to these changes and creates new fighter instances based on player name and chosen character
- Sent text differentiates Host and Client, which are assigned to the objects Player 1 and Player 2 respectively throughout the game
- Example sendText: ssm.sendText(“Host,”+state.player1.name+”Scorpion”);
    - Format: (Capitalized Player Type, + Player Name, + Chosen Character);

## Game View

- SSM sends socketed text about the state of the player – for example, if they are punching, moving, jumping, etc. so that it can update on the other player’s screen
- Based on the actions and keys that the press throughout the game, their actions change
- The listener will update the actions, and the View logic will update the animation images based on their actions
- Hitbox logic updates the HP lost and an action such as “got punched” if the player gets hit which is also sent over socketed network to update the player’s stagger animation and health bar
- Player 1 = host, Player 2 = client
- Example sendText usage:
state.ssm.sendText("host,"+state.currentPlayer.currentX+","+state.currentPlayer.currentY+","+state.currentPlayer.isAttacking+","+
state.currentPlayer.currentAction“,”+state.currentPlayer.movementDisabled+","+state.cur

rentPlayer.fighter.HP","+state.currentPlayer.hasRun+","+state.currentPlayer.fighter.isSpecialBeingUsed+","+state.currentPlayer.isBlocking);
  - Format: playerType, x-pos, y-pos, isAttacking, currentAction, isMovementDisabled, playerHP, playerHasAttackRun, isPlayerUsingSpecialMove, isPlayerBlocking

- Projectile states also sent over by SSM as they have their own instances in the game
- For example, Sub-Zero's Ice Ball is an object that is independent of Sub-Zero's fighter object, although it is Sub-Zero's boolean properties that activates the ice ball
- iceBallX controls the x-movement of the IceBall, toRender determines if the IceBall should be drawn
- Example Usage: state.ssm.sendText("iceBall1,"+state.iceBall1.iceBallX+","+state.iceBall1.toRender);
  - 2 IceBall instances initialized in the case that both players pick Sub-Zero
  - iceBall1 and iceBall2 used to differentiate host and client, respectively.
  - Format: playerTypeIceBall, iceBall's x-pos, shouldIceBallBeDrawn

## Chat View

- SSM is used to send and receive text messages sent throughout the game and is then displayed on a ChatView
- The ChatController displays this text with the player's name and their message on the Chat Screen (when opened)
  - In addition, message history (past messages) is also stored and displayed whenever ChatView is opened
- Example Usage: state.ssm.sendText("chat,"+player.name+","+message)
  - Format: chat, name of player, message typed
  - "chat" used as the first component of the message to differentiate this SSM listener from the rest of the game