Code Highlights for Mega-Game

Key input

```
} else if (gameState == "game") {
  if (key == 'a' || key == 'd') {
    player.isMoving = true;
    player.storedMoveKey = key;
  }
  if (key == ' ' && jumpKeyWasPressed == false || key == 'w' && jumpKeyWasPressed == false) {
    player.isJumpingOnce = true;
    jumpKeyWasPressed = true;
  }
```

In our program, key input itself is not tied directly to the moving, attacking or healing of the player character. Rather, input allows for booleans to change status which is then put to use in the continuous update-functions for the player character. This allows us to more precisely control order of updates and avoid problems of keyPressing activating again when button is held.

Highscore reading, sorting and writing

```
void read() {
  int i=0;
  for (TableRow row : table.rows()) {
    if (row.getInt("score") == 0 && row.getString("name") == null) {
      this.scores[i] = 0;
      this.players[i] = "AAA";
    } else {
      this.scores[i] = row.getInt("score");
      this.players[i] = row.getString("name");
    }
    i++;
  }
  println(scores);
}
```

```
void scoreSort() {
  boolean isSorted = false;
  while (!isSorted) {
    isSorted = true;
    for (int i=1; i<scores.length; i++) {
      if (scores[i-1]<scores[i]) {
        isSorted = false;

        int temp = scores[i];
        scores[i] = scores[i-1];
        scores[i-1] = temp;
        String tempStr = players[i];
        players[i] = players[i-1];
        players[i-1] = tempStr;
      }
    }
  }
```

The highscore function takes in any integer score and player name from an external list, adds any new score and sorts this incremented array in order of highest to lowest by comparing ever slot with the slot next to it. Thereafter, it proceeds to save the score in a separate file so it can be updated and reloaded across all uses of the program.

Enemy objects

```
//Enemy Parent Class
class Enemy {
  int hp, soul_leak; //Enemies have hitpoints and leak an amount of soul to the player on being hit
  int hitTimer = 0, hitTimerDuration = 10; //when can enemy be hit again
  boolean displayingEnemyWasHit = false; //for displaying red tint on enemy
  PVector pos, vel;
  float w, h;

  void takeDamage() {
```

All enemy type classes inherit data of parent Enemy class. Thus, we are able to handle all enemies at once by adding them to the same array of Enemy objects in the main program enemies [] and from there checking for player collision and the likes with the same lines of code whether there is one enemy or 10 different enemy types on the screen.