Design Documents

Designing Explanation

For the class diagram, every object is associated with the player in some sort of relationship. In this way the player is considered a controller. As we can see, every object is connected to the player object. If the player object doesn't exist, then most of the objects doesn't do anything. The only object that can be considered the exception is the title screen, and logo object. There is a relationship between these two, but it can exist without the player object. We can open the video game, and just open the title screen. It won't do anything, but we can exit the game easily without actually playing the game.

As I said earlier, almost all the objects in the class diagram cannot exist without the player object. We can see that most of the relationships with the objects are composition. They cannot technically exist without the player actually interacting with them. For example, the pause object will not do anything unless the player presses the escape key. The pause object won't freeze the game in the title screen, it would only do so when the player object exists. We can go on with the gameover object. It is only activated when the player object falls off the stage. The checkpoint is only changed when the player object collides with it. The room is another special case, the player object spawns there, but the room can exist without the player object. You can make a room, with many traps, but other than that, the room is only used to make sure the player doesn't fall off screen.

Because of this the class diagram is high cohesion, and high coupling due to the high tight the diagram is. Because of this, the code will be hard to reuse, because a video game is a very specific type of program. A database for alcohol inventory would be more manageable and easier to maintain then a video game. However, because it is a video game, this cannot be helped.

We can see this from looking at the sequence diagrams below. Like how the class diagram shows it as well, we can see that almost all use cases depends on the player object. Without the player object, none of the other objects would be necessary. The only objects that aren't dependent are the login, account, and title screen objects. Every other object wholly depends on the player object. Which shows why it is the controller. We can see with move, the use case needs player input. Same with jump, pause, dash, and chest. Chest, spike, and checkpoint needs the player object to collide with the objects to actually be of use. Game over, pause, title screen, and online scoreboard doesn't need the player object to collide with it, however they are direct consequences of the player's actions, and the user's input.

Now while it is true that it is hard to maintain. I will also point out that the player object doesn't create the objects. They are directly interconnected with many of them called in the player object using global variables and such. We can see from the sequence diagrams, that the player doesn't technically create the objects. The player object does call on it, but the objects

themselves can exist, but cannot be called upon without the player object. It is a very close relationship that cannot exist without the player object. That is why the player object is the controller, as it delegates through everything. With a press of a key, an object is called to affect the system in some way or another. We can see this clearly through the sequence diagrams, and the class diagrams below.

As for the next grasp pattern found in the class diagram below, we have a creator with the login and account objects. Without an account, the login object will not work. So, we can say that account has an initializing data for login. We can see that the relationship between the two is high coupling. Login is completely dependent on the account object to be able to create the account. Without it, login will not be able to go to the title screen. The game will never start, and the user will never be able to play the game. We can also say that account has a low cohesion as well, seeing as stated above, it creates the file, username, and password for the game. We can even look at the sequence diagram for the login and account creation use case to see how much the login depends on account object.

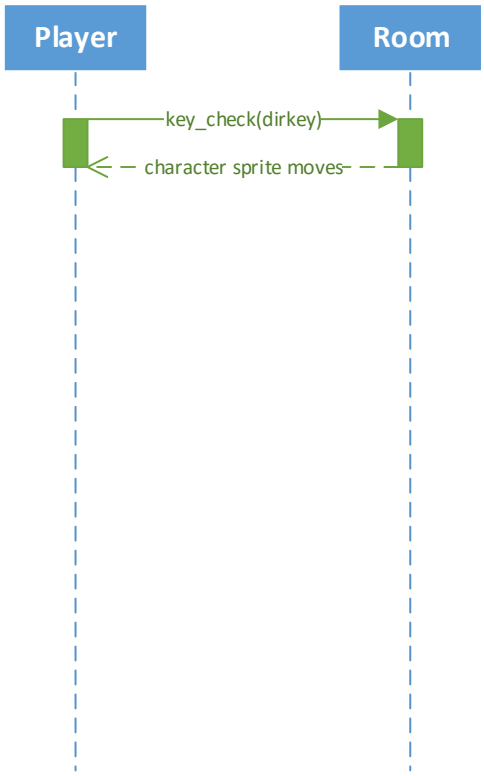
This is the strange part though. Because this is a game, generally a login and password is used for multiplayer games with online functionality. Our game isn't a multiplayer game. It is a single player game. However, hopefully in later iterations, rather than using the login credentials as a login, we can use it as a way to create save files, seeing as the way I created the login and account use case was to use the same methods to write a save file, it can be done as such.

We can also see that the game over object is also high coupling and low cohesion. The score upload and getting data from the website depends on the how the user interacts with the game over screen. We can say the same about the pause menu and the title screen. However, the game over object has a lot of functions implemented in it. I did this for several reasons, one, I believe from a gamer's standpoint, you would want to upload your high score right after you die. If you restart the level, your score goes back to zero. I also thought that you would like to see the top five scores, or you would like to visit the website in question. So, I added those functions to the game over object, because of how games are usually designed.

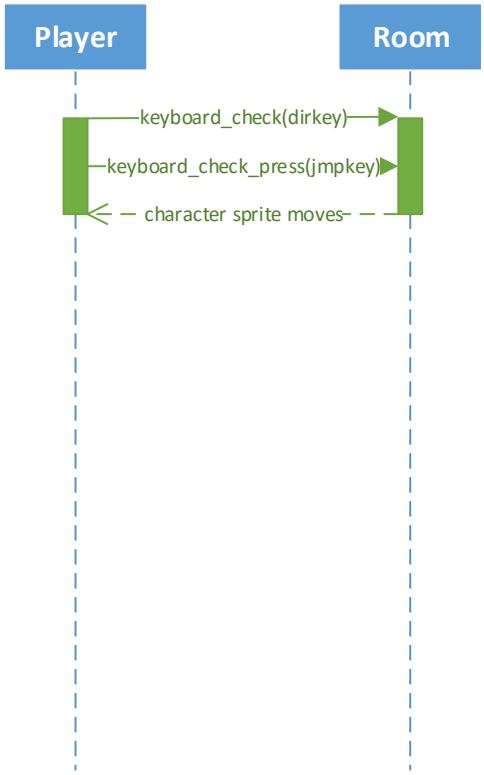
In conclusion, we can assume from the class diagram and the sequence diagrams that the player object is the controller. The other object relies on the player object to delegate when to use the objects in the system. This is done with keyboard or video game controller inputs from the actual player. As for the login and account creation use case, I would say that they are creators, account creates the login file for the login object.

Sequence Diagrams

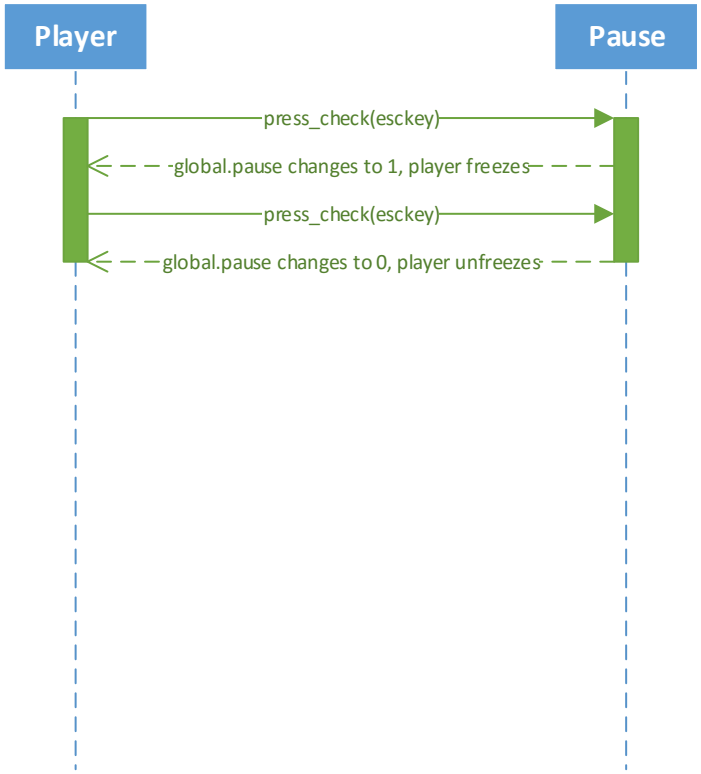
1) **Move –**



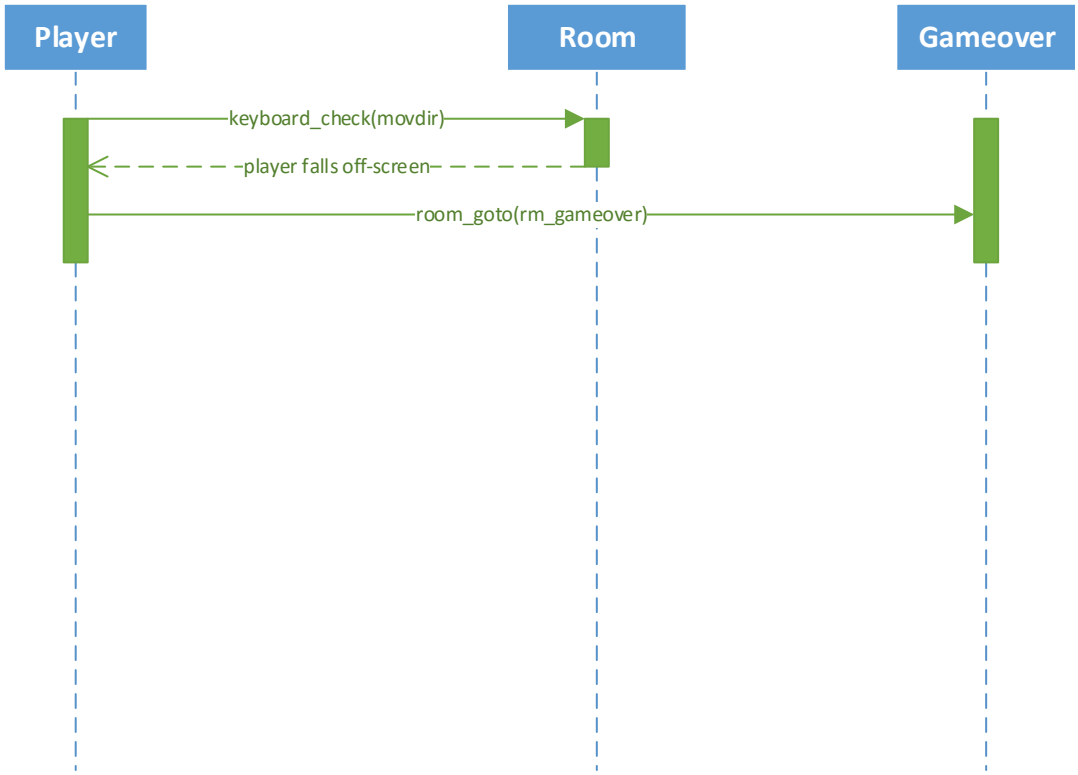
2) **Jump –**


Player
Room
keyboard_check(dirkey)
keyboard_check_press(jmpkey)
character sprite moves

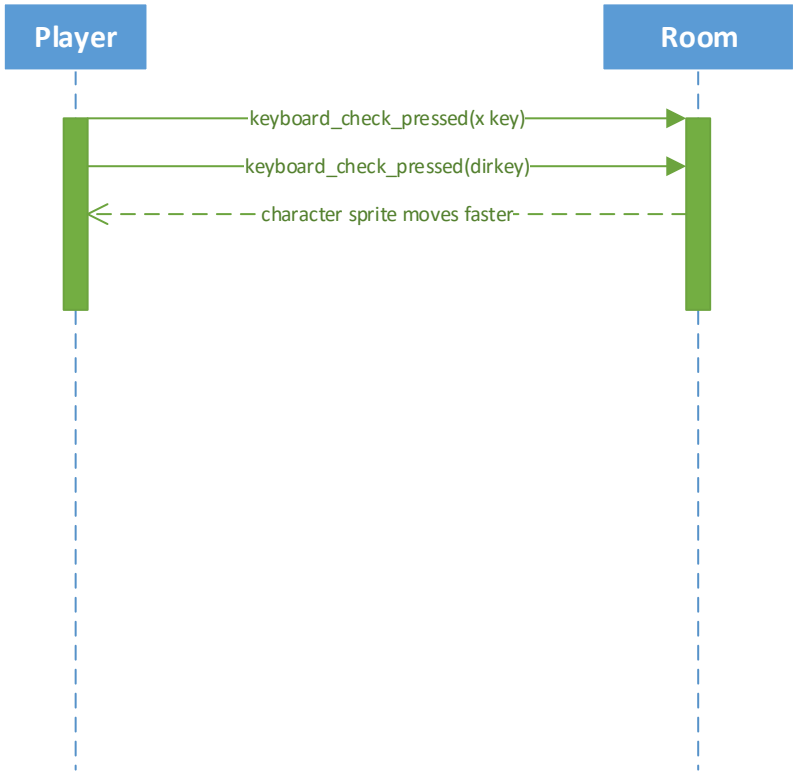
3) Pause –


Player
Pause
press_check(esckey)
global.pause changes to 1, player freezes
press_check(esckey)
global.pause changes to 0, player unfreezes

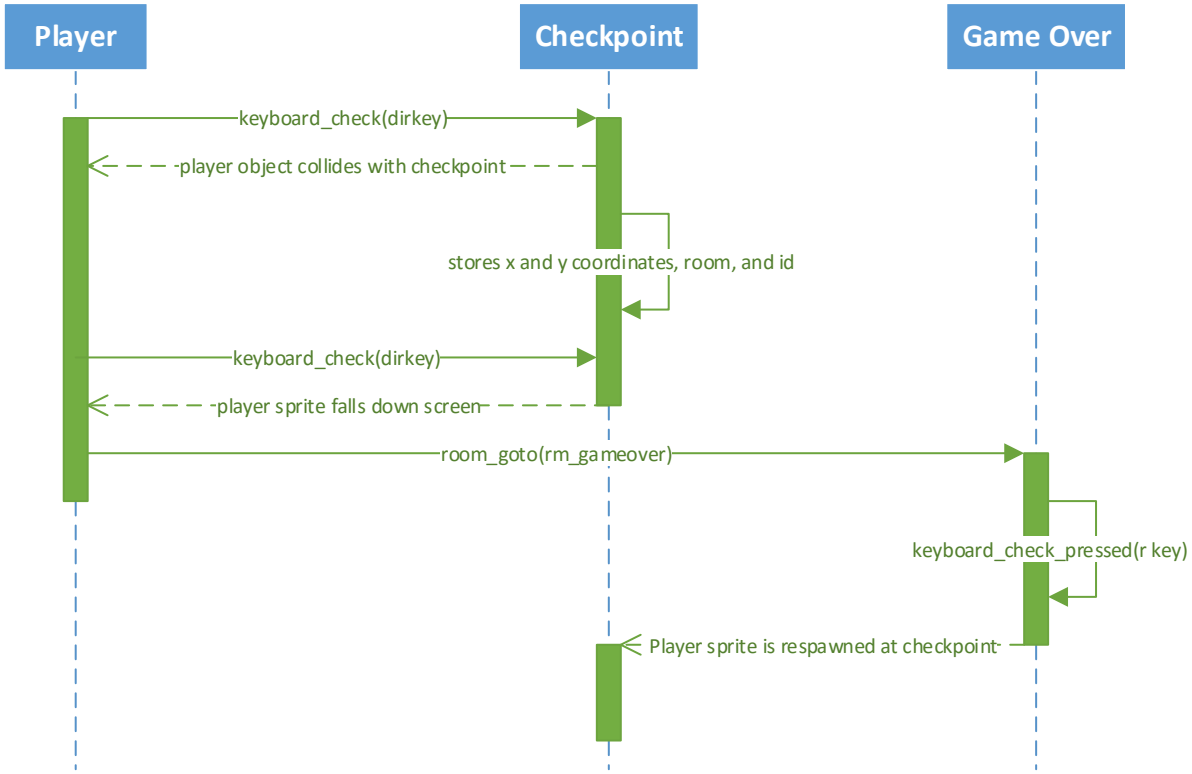
4) **Gameover –**



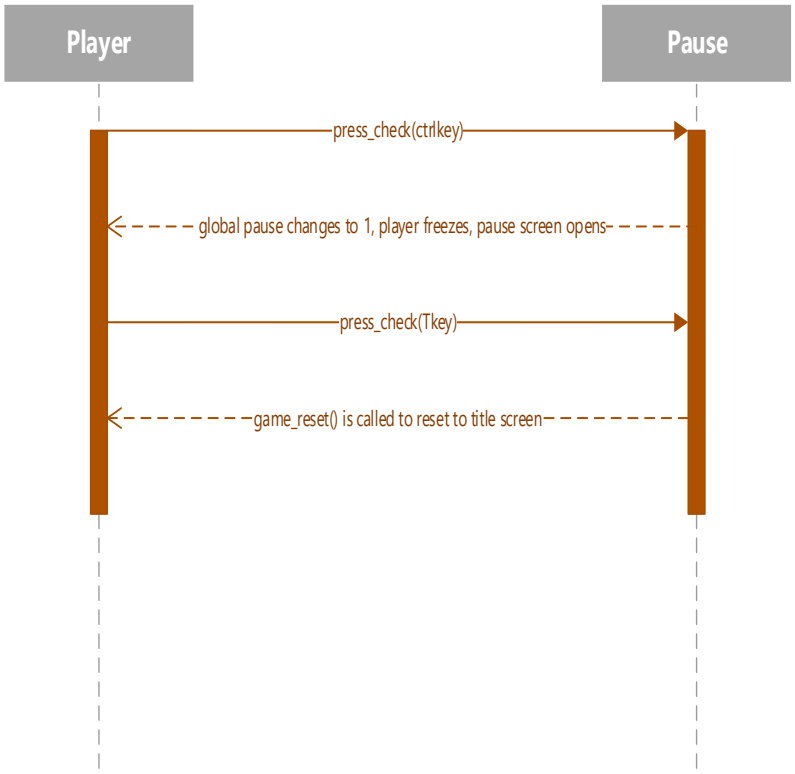
5) **Dash –**



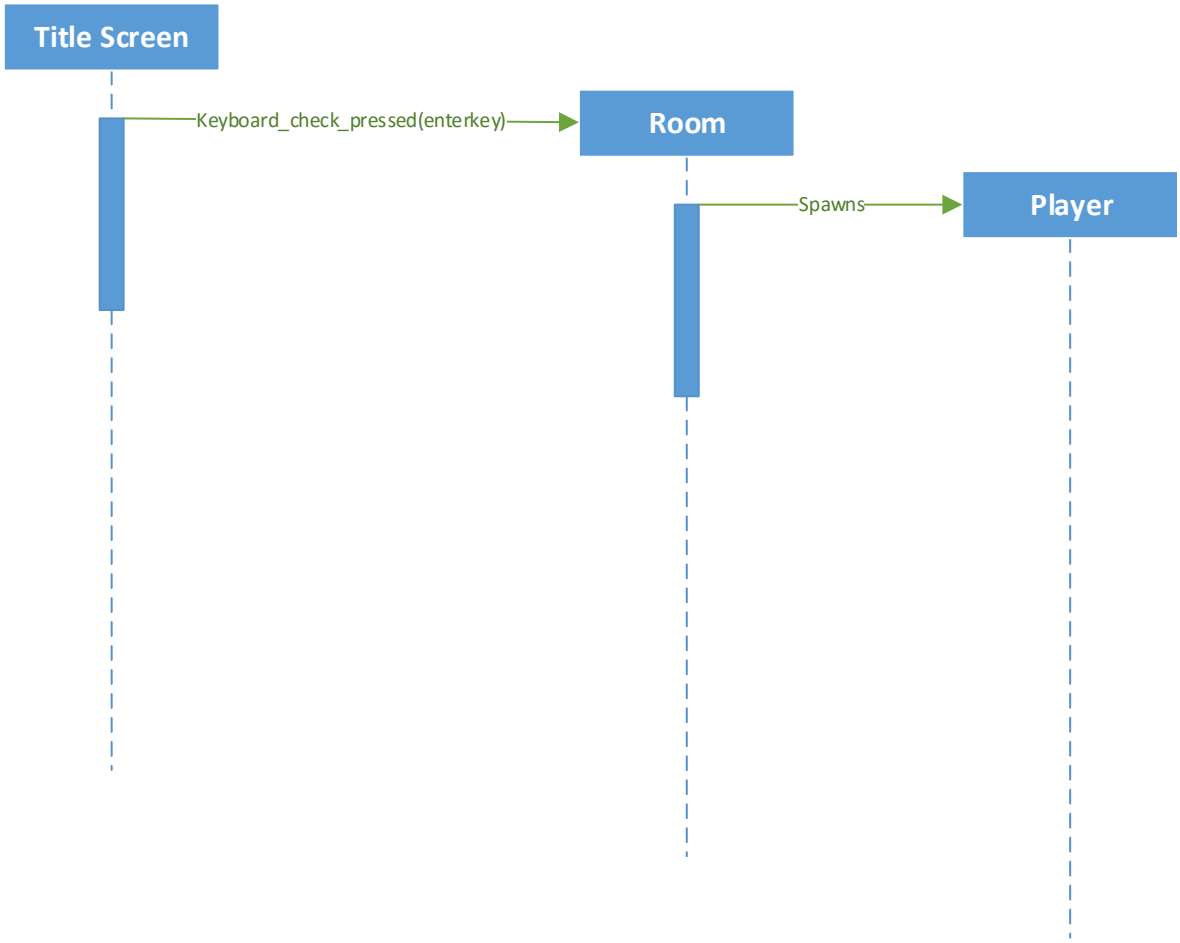
6) Checkpoint –



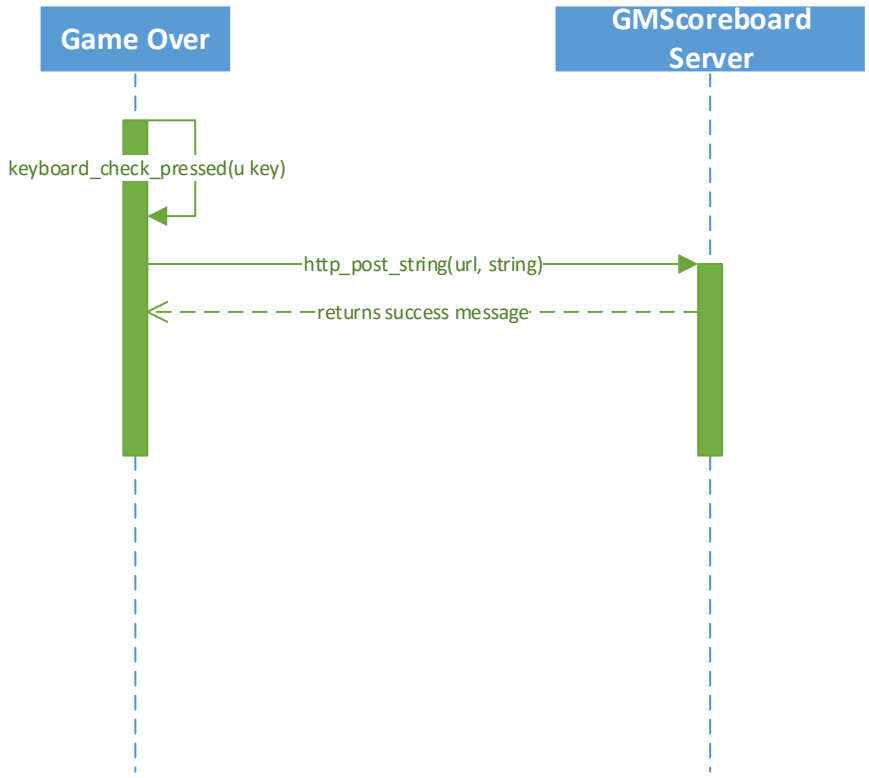
7) Pause Menu –



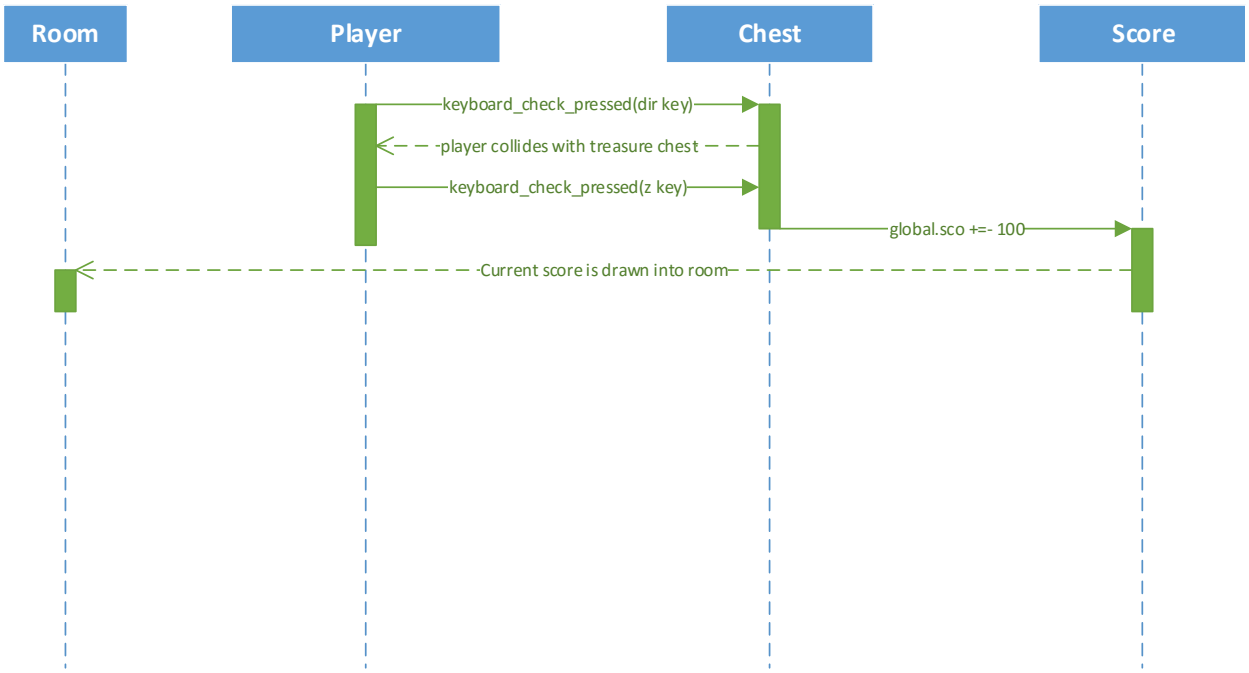
8) Title Screen –



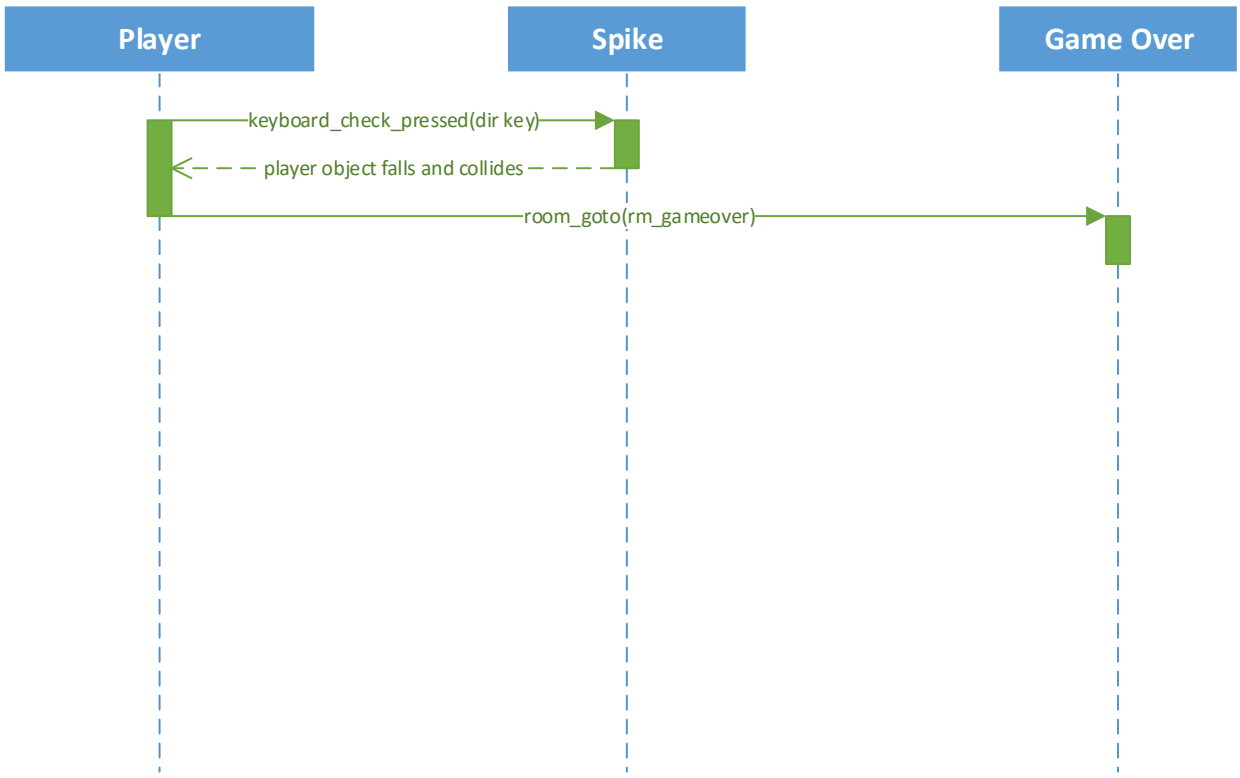
9) Online Scoreboard –



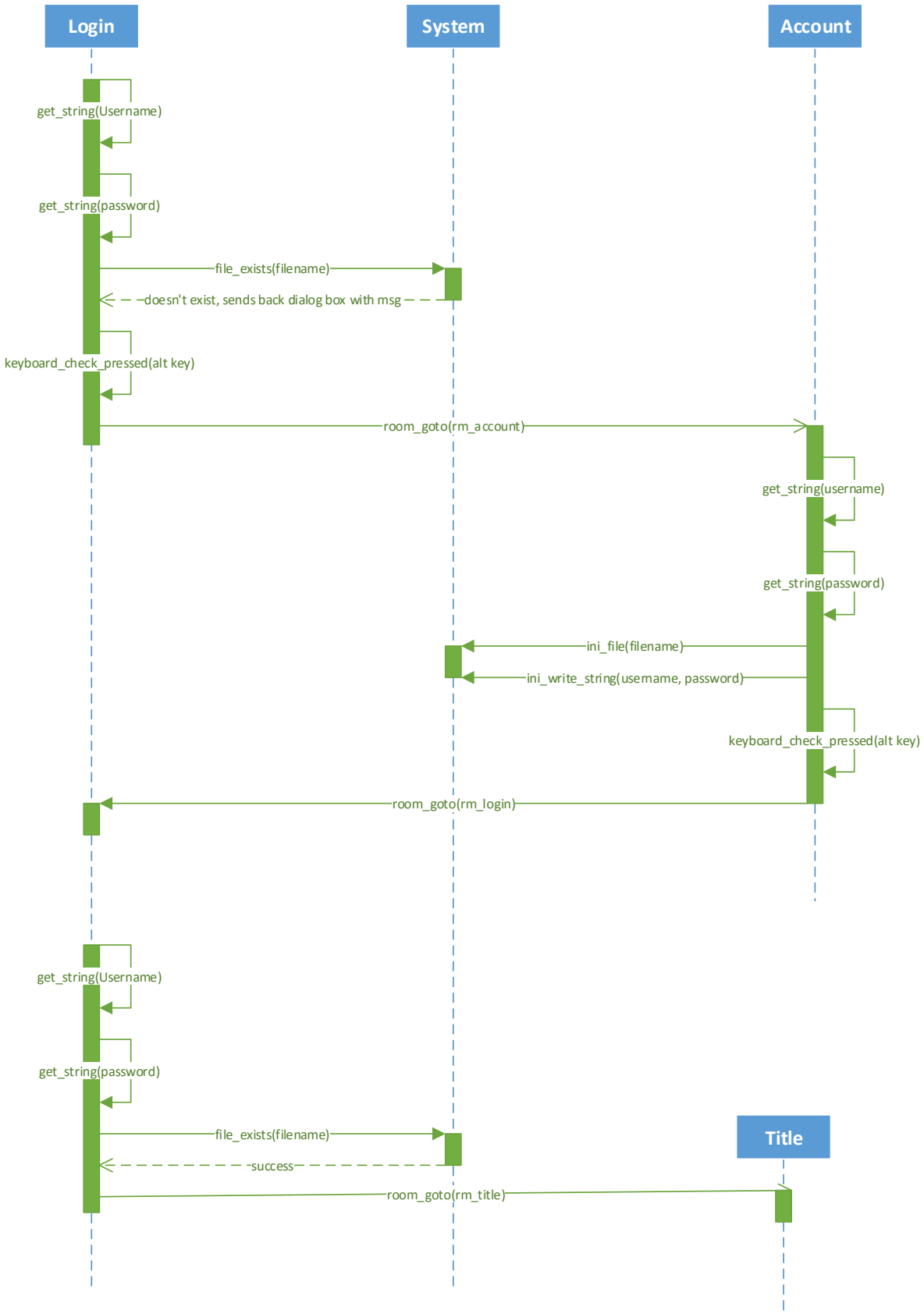
10) Treasure Chest –



11) Spike –



12) Login & Account Creation –


Login
System
Account
get_string(Username)
get_string(password)
file_exists(filename)
doesn't exist, sends back dialog box with msg
keyboard_check_pressed(alt key)
room_goto(rm_account)
get_string(username)
get_string(password)
ini_file(filename)
ini_write_string(username, password)
keyboard_check_pressed(alt key)
room_goto(rm_login)
get_string(Username)
get_string(password)
file_exists(filename)
success
Title
room_goto(rm_title)

Class Diagram