Change made from UML design:

- Adding many constants which to the class which were not defined in the original UML. EG: ONESECONDS, ZEROPOINT4SECONDS in the World. I did not realize there were so many constant after actually implement the code
- Adding a collisionEffect to all object. This method is called when Player made contact with it. This is a polymorphism that in coding I can write less if statement when player made contact with other obejcts.

World Class:

- Add Many methods that can change sprites. This is because all sprites object are object in the World. Therefore, if we want to change object in the World, we should notify world class.

Sprite Class:

- Delete tempSprite[][]][from Sprite Class, it turns out that using tempSprite[]][][] as buffer for temporary is meaningless, I could just do the sprite updating in the World. And it makes more sense, because in my design, every change in the sprite should notify world because they are all objects in the world.
- Store game coordinate rather than screen coordinate in each sprite. As game Coordinate works better when we call update the game. We only change game coordinate to screen coordinate when we render the Sprite.
- Adding many reference to Npc, switch. As they will be called during different class's update method, therefore it will be better to notify world class that they will be changed when we update a specific sprite.
- Change sprites to a three dimension array with the lowest dimension to be levels on one Sprite. This means on every location, there are three levels. The bottom level is floor, mid-level is switch, target, door and the rest are all in top levels. This is because I found out in the mapfile, target and floor are at the same location.

NPC Class and its subclass:

- Remove nxtPos() method as it turns out that we don't need it

Block ClasS:

- Remove canMove Method. Instead, implement boxCanMove for different blocks.

Tile Class:

- Remove canGOTHROUGH attribute, rather, defined a isBlocked function in the World class. Because it turns out for different object, they have different definition that whether a sprite can go through.

ICE class and Render Class:

- OverWriting render method to support explosion and sliding effect.

Door class:

- Remove switch Position as door do not need to know where is switch.
- Add isOpen method to return whether this door is open

Any Difficultis you had during this Project:

- Time limit. Though I drawe the UML first, I did not go through the logic about how update changed different class's attributed. During that time, I throwed OOP idea away and just changed instance attribute whenever I want and did not care about privacy. This result in complete logic confusion at the beginning, and at some stage I had to just rewrite everything. In which case I then wrote a sudo code for update method, consider the input and output for each class, set up an order for render. Then I start coding. But it was very close to the deadline, thought I finished basic specification in the specification, if I had more time, I could make the project more perfect.

One Key Piece of Knowledge you have Learned:

- From the project I have deeper understanding in polymorphism and inheritance. For instance, in the Project 1, I did not know whether child class method can change the instance variable which defined in the parent class. This caused me a lot of trouble in Project1, in which case I had to use a very weird logic to update my Player position, and tutor comment that "the logic in updating player is a bit weird". However, this time, I learned that if a child class's method want to change the variable that inherited from parent class, it should call the parent class's setter to set it. In this way I could maintain the oop idea, while do the job.

  Speak of Polymorphism, I was not sure when I have an parent class reference to a object, which actually was created as a child class object, how should I use child Class method. One way is to perform overriding in the child class, therefore when the object is called by the parent class reference, it will perform the lowest level of class's method.

Anything you would do differently if you did a similar Program:

- Instead of Just drawing the UML, I should also consider the doing sudo code for some key methods. For example, the update method in the Program. The logic was quite confusing in my first attempt to do the project2B. I throwed away all OOP design and just tried to get everything work. And after a few days, I could not recognize my code. Such rush to finish a function did not speed up my work, rather, I have to delete everything, and reconstruct my UML and Logic in update methods to make my work more clear, as well as setting up a clear logic level. Which, in the afterward work, I keep stick to the OOP idea as much as possible,

and this saves me a lot of time because now I can treat many functions as a black box – all because I strictly consider the logic and what input, out should be. I wasted a lot of time to reconstruct my rush code, next time I should plan carefully first, then do the code. From this project, I have enhanced my understanding in the importance of design and OOP idea. Design will not waste your time, but provide clear logic and efficiency.