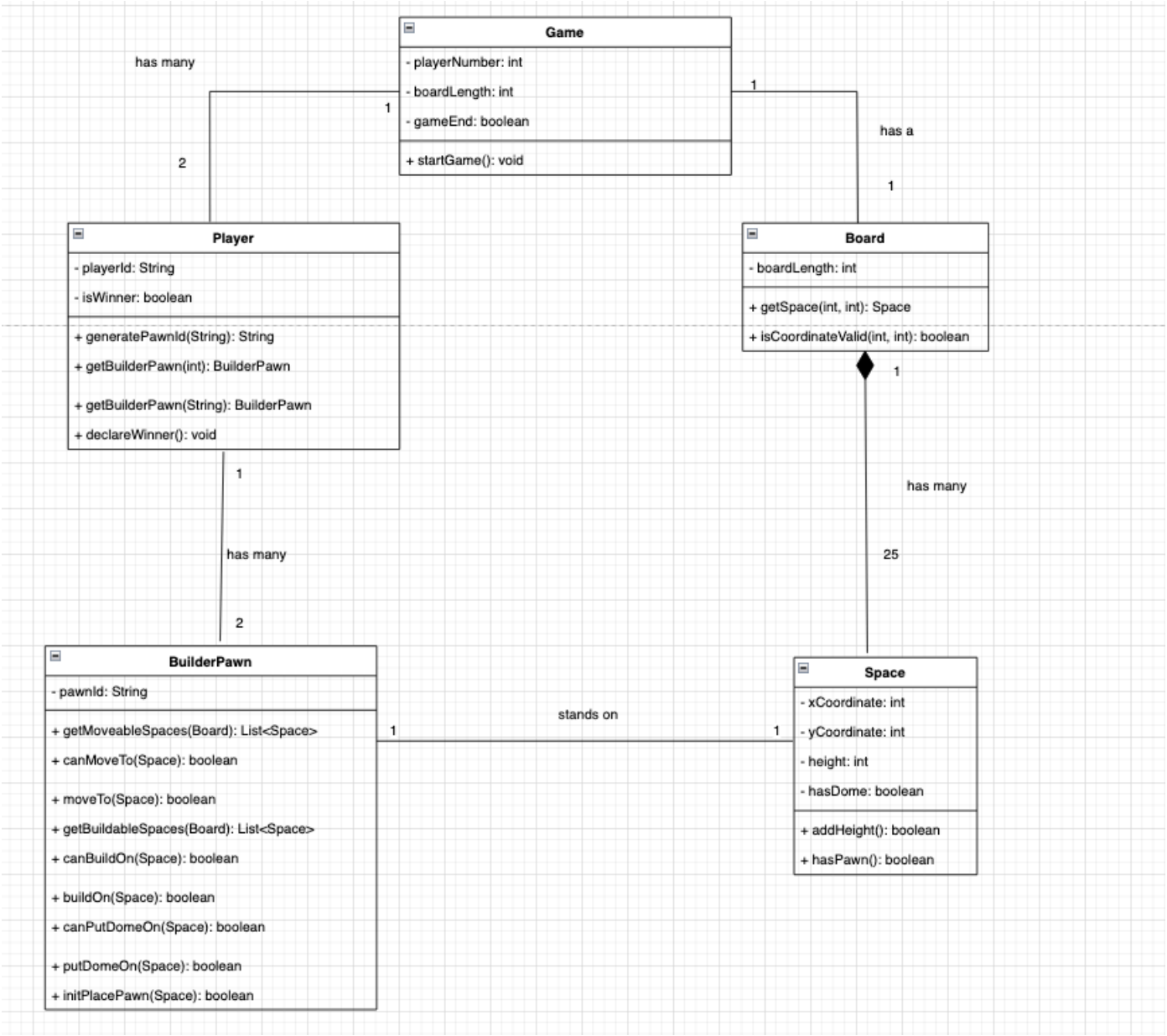
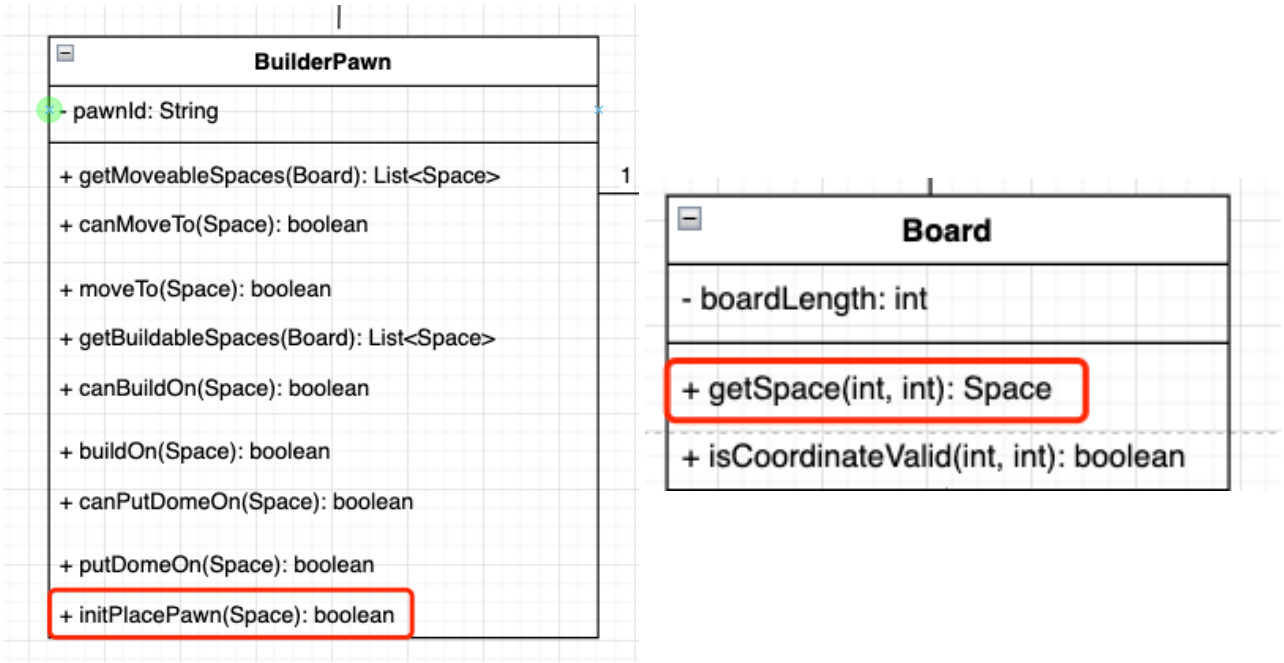
0. Object Model Overview



1. Player Interactions(Actions)

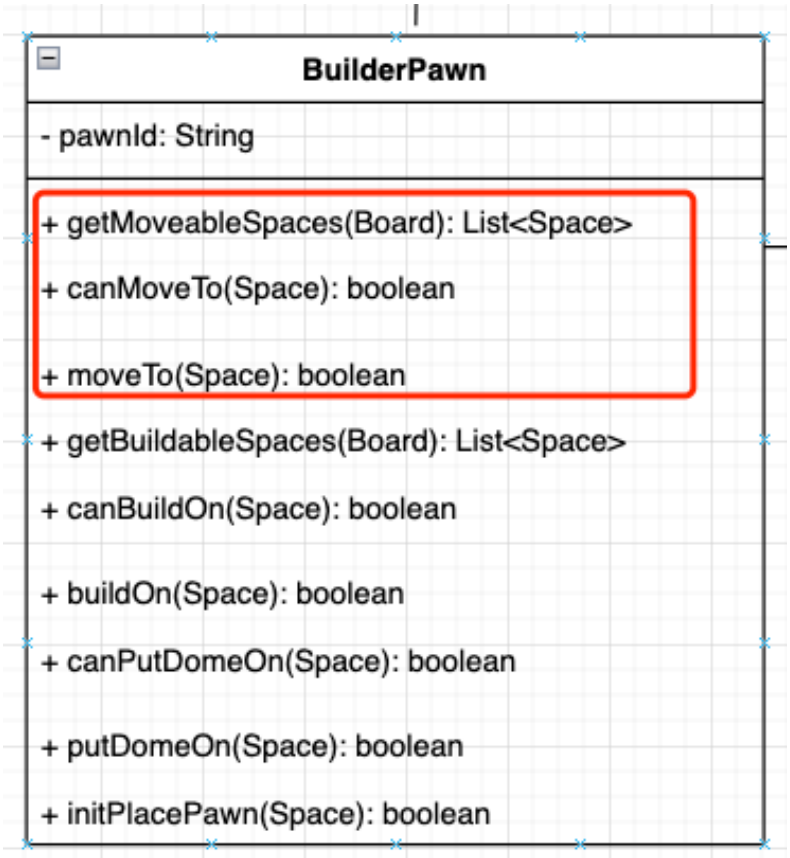
Note: Most of the actions including move, build are delegated to *BuilderPawn* Class in my design.

Initially place pawns



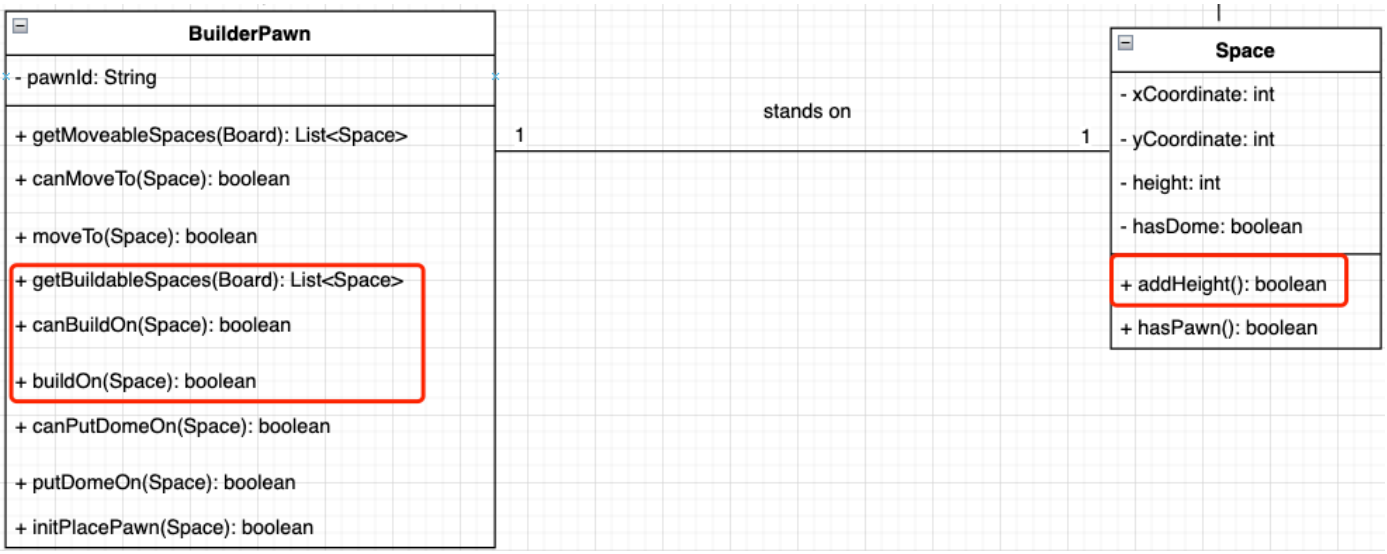
1. When the player wants to initialize one of its builderpawn on the board, he should give the X and Y coordinate to go.
2. The ***getSpace(x, y)*** function will be called to find the corresponding ***Space*** instance on board.
3. The ***initPlacePawn(Space)*** method will be called by the ***buildPawn*** instance.
    - If the function returns true, the placement is successful and can move to next step.
    - If the function returns false, it indicates the placement failed possibly because the place is already taken by another pawn. The player would have to choose another space.

Move a Pawn



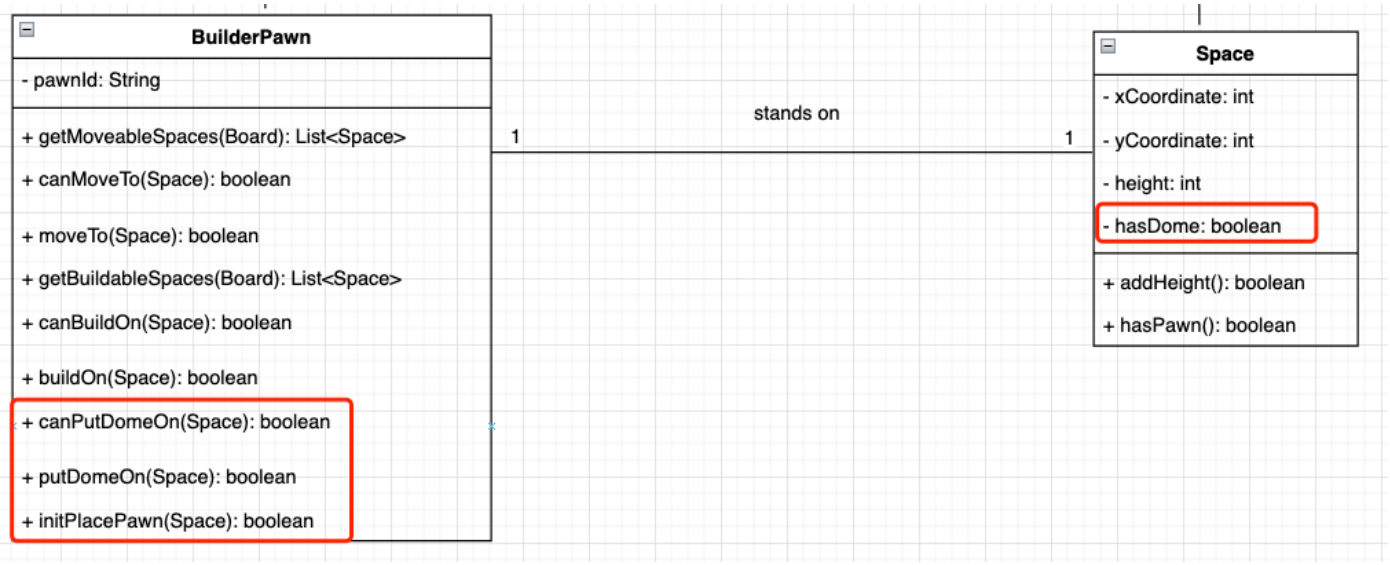
1. The player wants to move one of its pawn, the ***getMoveableSpaces()*** function will be called and returns a list of spaces where the pawn ***canMoveTo()***.
2. Then the player should choose a target space and pass the space to ***moveTo()*** function.
3. The ***moveTo(space)*** function will
    - break the association between current space and pawn.
    - build new association between new target space and pawn.
    - If the move can satisfy game end condition, it will set the player's winner flag and game's end flag by calling ***player.declareWinner()***.

Build



1. The player wants to build, the ***getBuildableSpaces()*** function will be called and returns a list of spaces where the pawn ***canBuildOn()***.
2. Then the player should choose a target space and pass the space to ***buildOn()*** function.
3. The ***buildOn(space)*** function will call ***space.addHeight()*** to add a new level to the space.
    - If the space is already level 3 or has a dome on it, it will return FALSE to indicate the build failed.
    - Otherwise the height will increment and return TRUE.

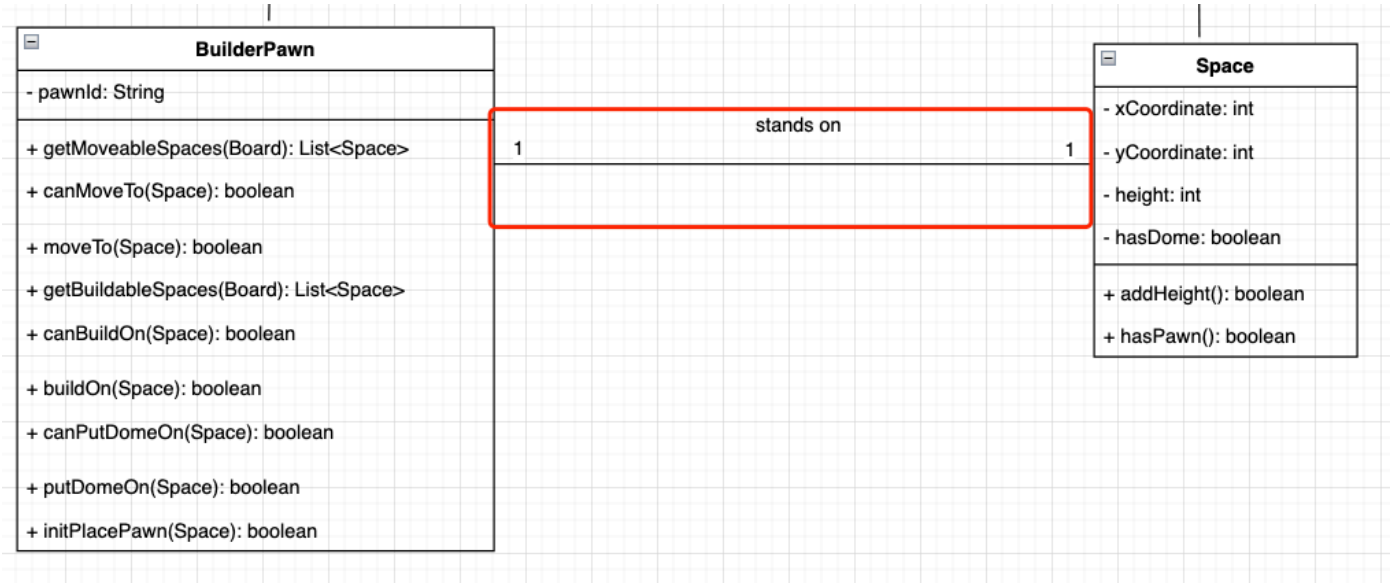
Put Dome on Space



1. The player wants to build dome on the space, the ***getBuildableSpaces()*** function will be called and returns a list of spaces where the pawn ***canPutDomeOn()***.
2. Then the player should choose a target space and pass the space to ***putDomeOn()*** function.
3. The ***putDomeOn(space)*** function will call ***space.setHasDome(true)*** to add a dome to the space.
    - If the space is smaller than level 3 or already has a dome on it, it will return FALSE to indicate the build failed.
    - Otherwise the hasDome will be set to TRUE and return TRUE.

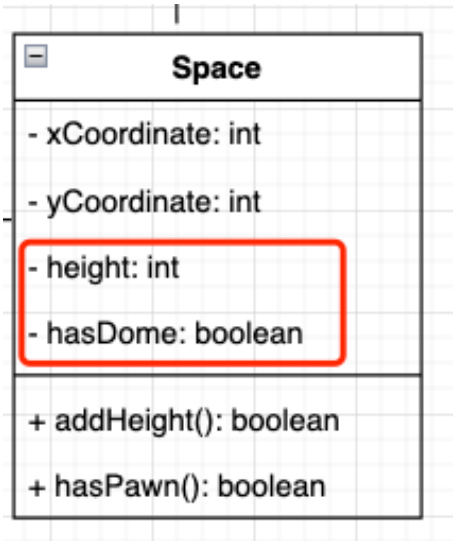
2. Game State Storage

Pawn Position State



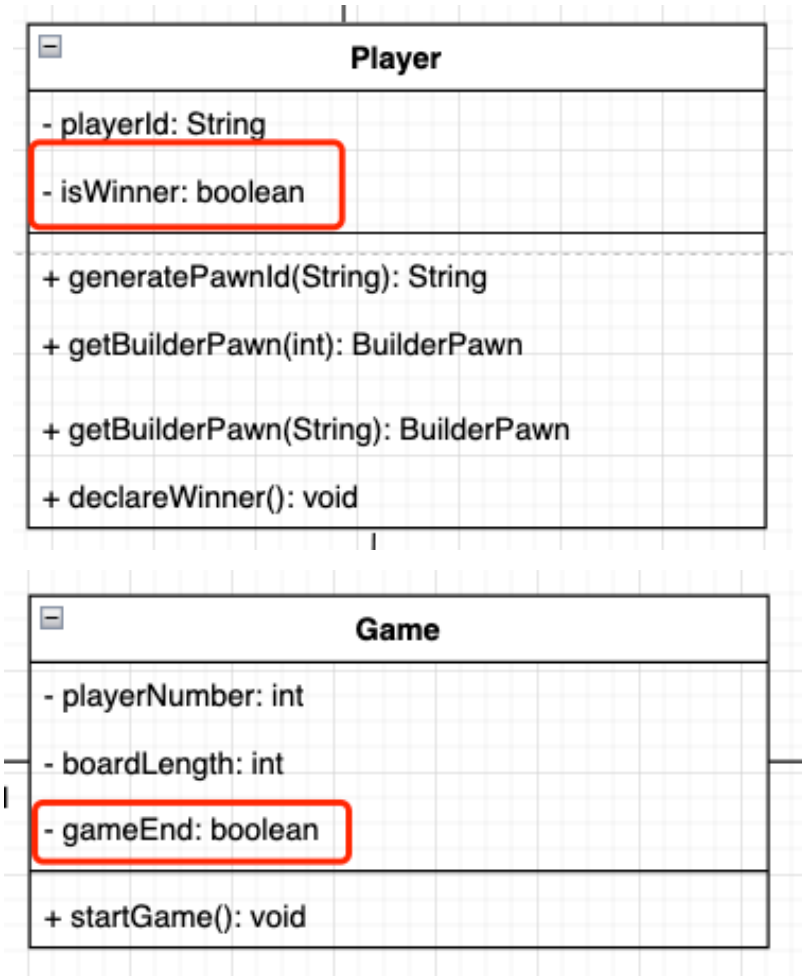
We need to store the position of each pawn at every moment of the game. We achieve that by making them associated. Once the state changes, both of them would change/break the association.

Space Tower Level State



We store the Tower level state information in the ***Space*** Object with the attribute ***height*** and ***hasDome***.

Is Game Ended and Who is the Winner

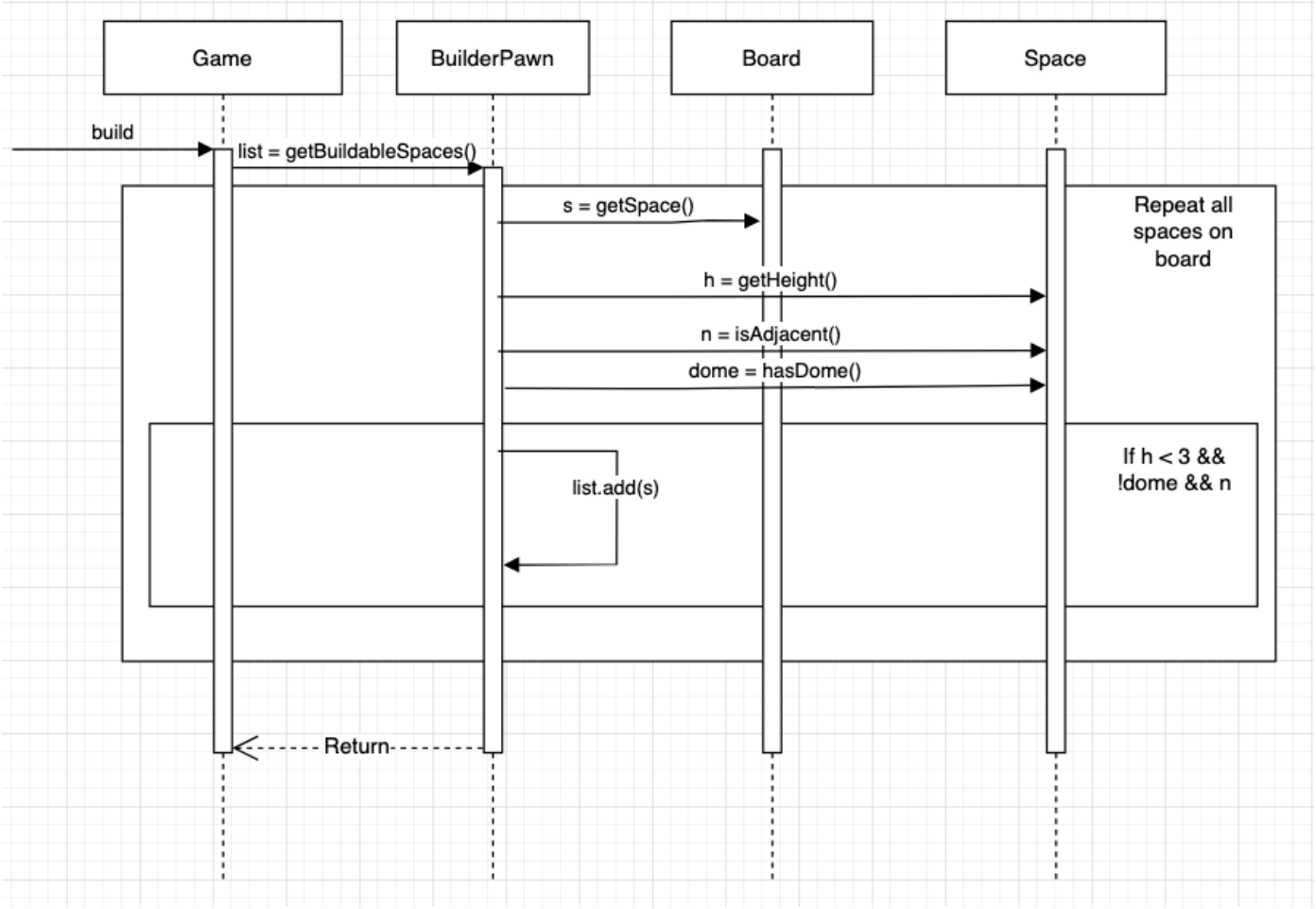


The winner information is stored in a boolean attribute ***isWinner***.

The game end flag is store in boolean attribute ***gameEnd***.

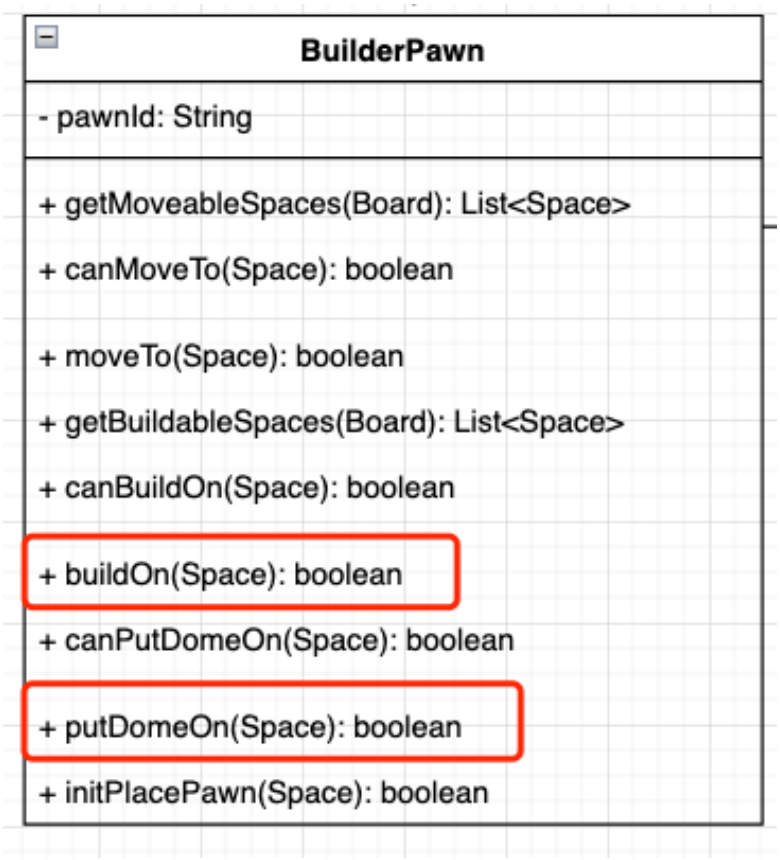
3. Build Detail

Check if Build is Valid

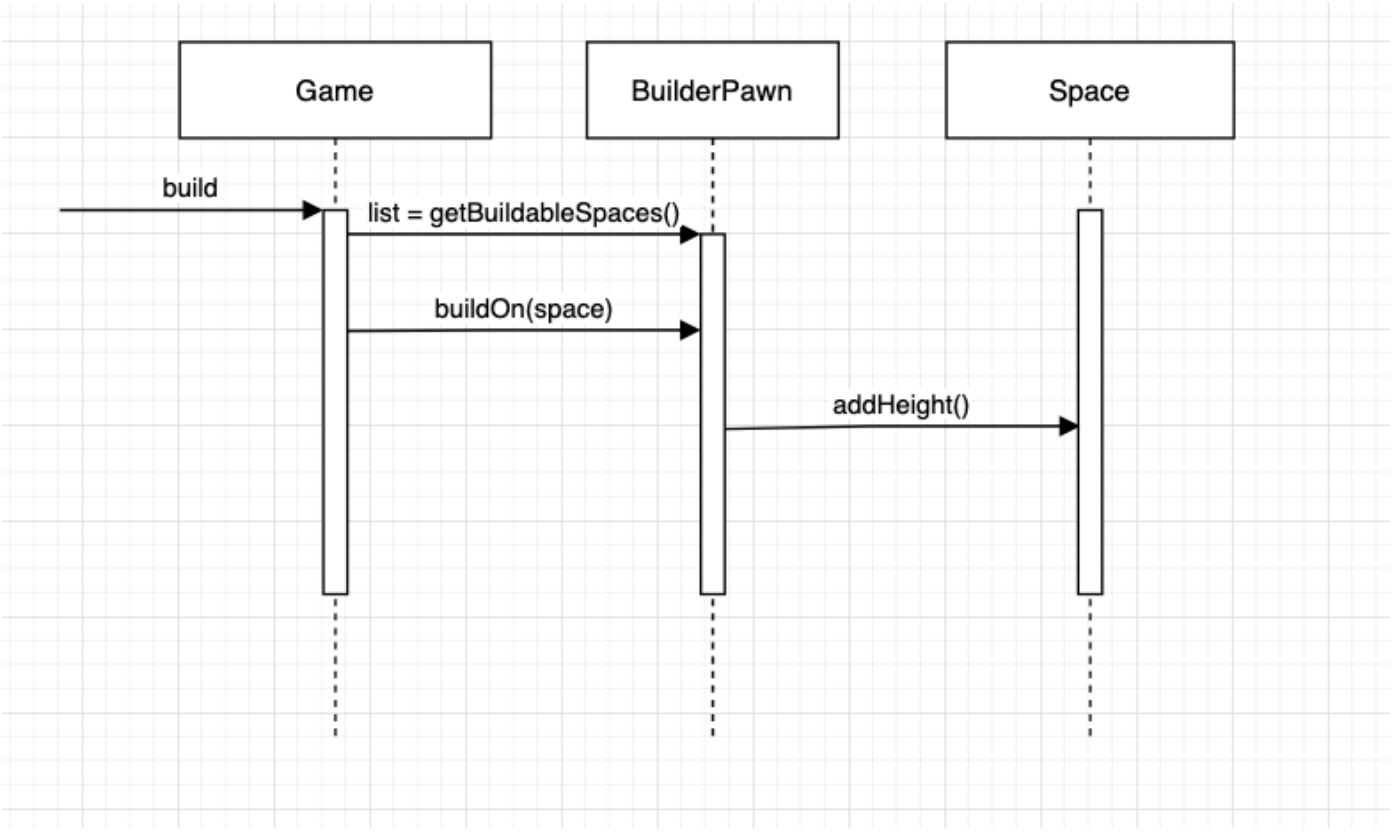


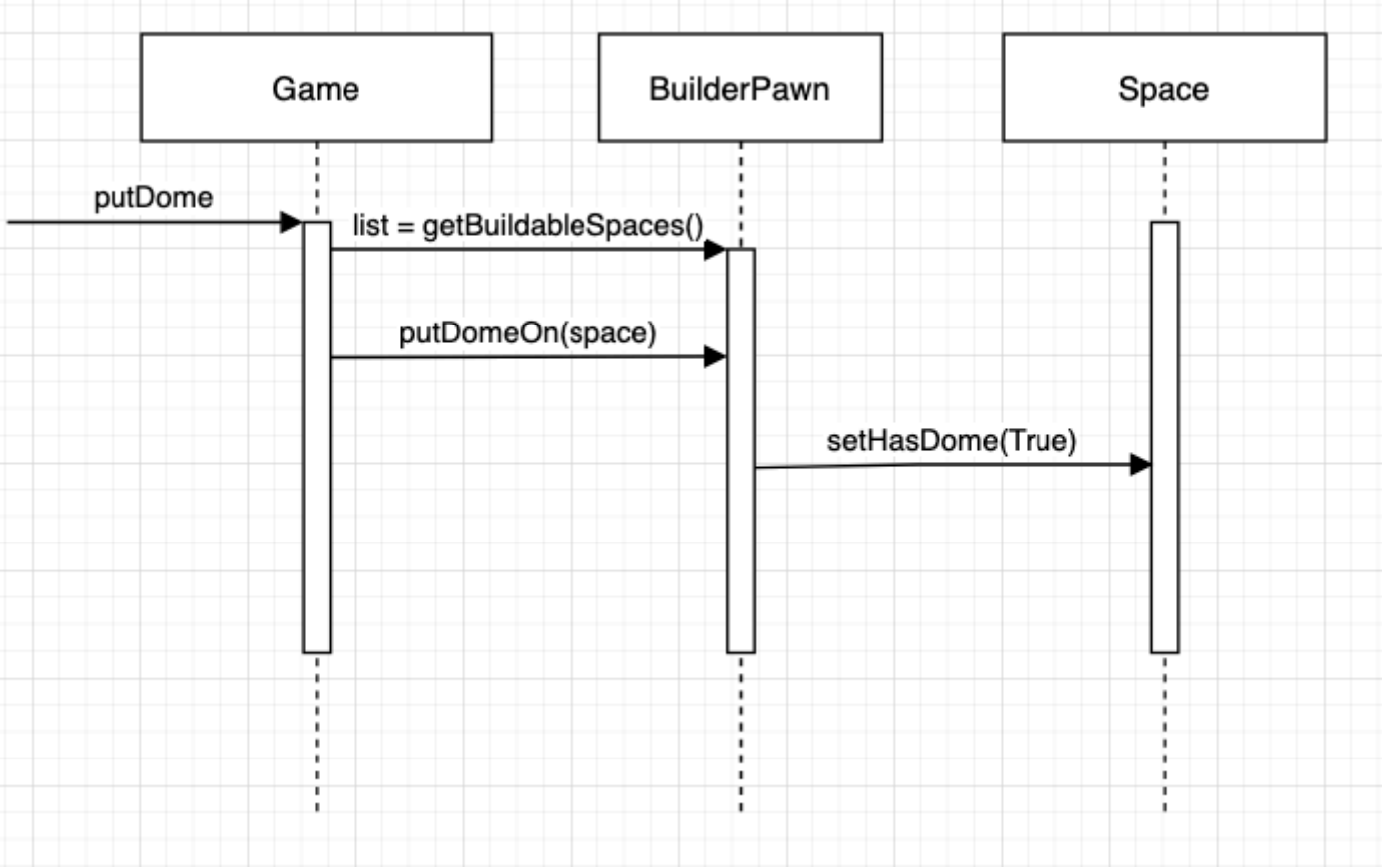
This is my interaction diagram for telling if the build is valid. Basically the game will call the ***getBuildableSpaces()*** method of the builderpawn, which returns a list of space objects that the pawn could build on. The space must be adjacent, has no dome and height smaller than 3.

Note: I have splitted the action of ***Build*** and ***putDomeOn*** to 2 differnt actions. The user will have to choose the specific interation type instead of calling build and automatically adding a dome to the tower. Players will have to call ***putDomeOn*** explicitly, as the methods in diagram shown below.



Perform Build





The game will perform build by calling the ***buildOn(Space)*** method of the builder pawn. Then it will call the ***addHeight()*** method of the space instance.

Similarly, the put dome action will be done by calling ***putDomeOn()*** method, and then set the hasDome attribute in the space instance to true.

The related object model is as below: