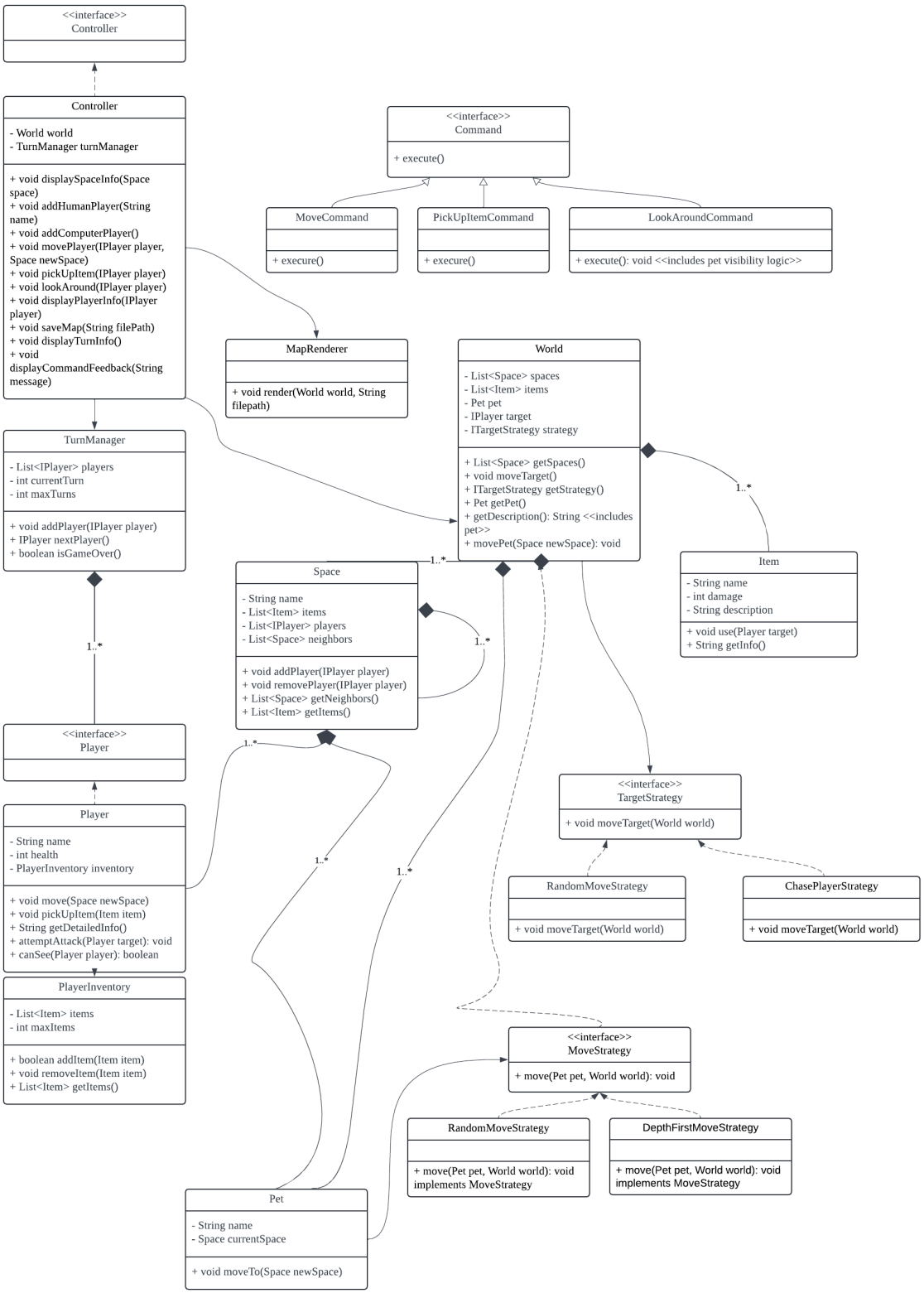
1. UML


<<interface>>
Controller
Controller
- World world
- TurnManager turnManager
+ void displaySpaceInfo(Space space)
+ void addHumanPlayer(String name)
+ void addComputerPlayer()
+ void movePlayer(IPlayer player, Space newSpace)
+ void pickUpItem(IPlayer player)
+ void lookAround(IPlayer player)
+ void displayPlayerInfo(IPlayer player)
+ void saveMap(String filePath)
+ void displayTurnInfo()
+ void displayCommandFeedback(String message)
<<interface>>
Command
+ execute()
MoveCommand
+ execure()
PickUpItemCommand
+ execure()
LookAroundCommand
+ execute(): void <<includes pet visibility logic>>
MapRenderer
+ void render(World world, String filepath)
World
- List<Space> spaces
- List<Item> items
- Pet pet
- IPlayer target
- ITargetStrategy strategy
+ List<Space> getSpaces()
+ void moveTarget()
+ ITargetStrategy getStrategy()
+ Pet getPet()
+ getDescription(): String <<includes pet>>
+ movePet(Space newSpace): void
TurnManager
- List<IPlayer> players
- int currentTurn
- int maxTurns
+ void addPlayer(IPlayer player)
+ IPlayer nextPlayer()
+ boolean isGameOver()
Space
- String name
- List<Item> items
- List<IPlayer> players
- List<Space> neighbors
+ void addPlayer(IPlayer player)
+ void removePlayer(IPlayer player)
+ List<Space> getNeighbors()
+ List<Item> getItems()
Item
- String name
- int damage
- String description
+ void use(Player target)
+ String getInfo()
<<interface>>
Player
Player
- String name
- int health
- PlayerInventory inventory
+ void move(Space newSpace)
+ void pickUpItem(Item item)
+ String getDetailedInfo()
+ attemptAttack(Player target): void
+ canSee(Player player): boolean
PlayerInventory
- List<Item> items
- int maxItems
+ boolean addItem(Item item)
+ void removeItem(Item item)
+ List<Item> getItems()
<<interface>>
TargetStrategy
+ void moveTarget(World world)
RandomMoveStrategy
+ void moveTarget(World world)
ChasePlayerStrategy
+ void moveTarget(World world)
<<interface>>
MoveStrategy
+ move(Pet pet, World world): void
RandomMoveStrategy
+ move(Pet pet, World world): void implements MoveStrategy
DepthFirstMoveStrategy
+ move(Pet pet, World world): void implements MoveStrategy
Pet
- String name
- Space currentSpace
+ void moveTo(Space newSpace)
1..*

2. TEST PLAN

Class Name	Test Name	Test Condition	Input Examples	Expected Output
Player	attemptAttack(Player)	Target is visible and player has multiple weapons	Player1 in room A, Player2 in adjacent room B, weapon1 and weapon2 available	Attack succeeds, Player2 health reduced by weapon1 or weapon2 damage
Player	attemptAttack(Player)	Target visible but with insufficient health	Player1 attacks Player2 with more damage than Player2's health	Player2 health drops to 0, Player2 dies
Player	attemptAttack(Player)	Target out of view due to pet obstruction	Player1 in room A, Player2 in room C, pet in room B	Attack fails, returns "Target not in sight"
Player	canSee(Player)	Player moves between spaces, visibility recalculated	Player1 moves from room A to B, Player2 in adjacent room C	After movement, visibility recalculated, returns true or false accordingly
Player	canSee(Player)	Adjacent space visibility check with no pet	Player1 in room A, Player2 in room B, no pet	Returns true
LookAroundCommand	execute()	Player looks around when surrounded by multiple players	Player in room A, Player2 and Player3 in adjacent rooms	Player sees only neighboring rooms without players
LookAroundCommand	execute()	Player looks with pets in alternate neighboring spaces	Player in room A, pets in rooms B and D, no pet in room C	Player sees rooms A and C, but not B or D
LookAroundCommand	execute()	Player and pet in the same room, pet obstructs view	Player and pet both in room A	Only current space details are visible
MoveStrategy	move(Pet, World)	Random move in a space with multiple branching paths	Pet in room A, branches to rooms B, C, and D	Pet randomly moves to any of B, C, D
MoveStrategy	move(Pet, World)	Random move with no available spaces	Pet in a dead-end room A	Pet stays in room A

DepthFirstMoveStrategy	move(Pet, World)	Depth-first search involving revisiting previously visited spaces	Pet in room A, rooms B, C visited previously	Pet chooses a new unvisited path or revisits as per DFS rule
World	movePet(Space)	Moving pet when another player is already present	Move pet from room A to room B, Player1 is in room B	Move succeeds, pet and Player1 coexist in room B
World	movePet(Space)	Moving pet to its current space	Pet in room B, attempt move to room B	No change, returns "Already in the space"
World	movePet(Space)	Attempting to move pet in an occupied cycle of spaces	Cycle between rooms A, B, C with multiple players	Successfully finds next available space or fails gracefully
Pet	moveTo(Space)	Move to a connected space with multiple exits	Pet moves from room A with exits B and C	Pet moves successfully to a randomly chosen exit
Pet	moveTo(Space)	Attempt to move to a non-adjacent space	Pet in room A, trying to move to room D which is not connected	Throws InvalidSpaceException
Pet	moveTo(Space)	Move when obstructed by locked path	Path to room C is locked, Pet tries to move to room C	Fails, returns "Path is locked"
PlayerInventory	addItem(Item)	Add item beyond capacity with priority item present	Inventory full, add higher priority item	Successfully replaces an existing item based on priority
PlayerInventory	addItem(Item)	Add item to empty inventory	Inventory empty, adding item "Sword"	Successfully adds item "Sword"
PlayerInventory	removeItem(Item)	Remove item from inventory with duplicate items	Inventory contains two "Potion" items	Successfully removes one instance of "Potion"
PlayerInventory	removeItem(Item)	Remove non-existent item	Inventory does not contain "Shield"	Fails, returns "Item not found in inventory"
TurnManager	nextPlayer()	Switching turns after elimination of one player	Players: [Player1, Player2, Player3], Player2 eliminated, current: Player1	Next player is Player3
TurnManager	nextPlayer()	Cycling through with only one player left	Players: [Player1], current: Player1	Returns Player1, since only one player remains
TurnManager	isGameOver()	Game over after reaching maximum allowed turns	Maximum turns set to 10, current turn 10	Returns true, game ends

TurnManager	isGameOver()	Game not over with remaining players and turns	Players: [Player1, Player2], turns left > 0	Returns false, game continues