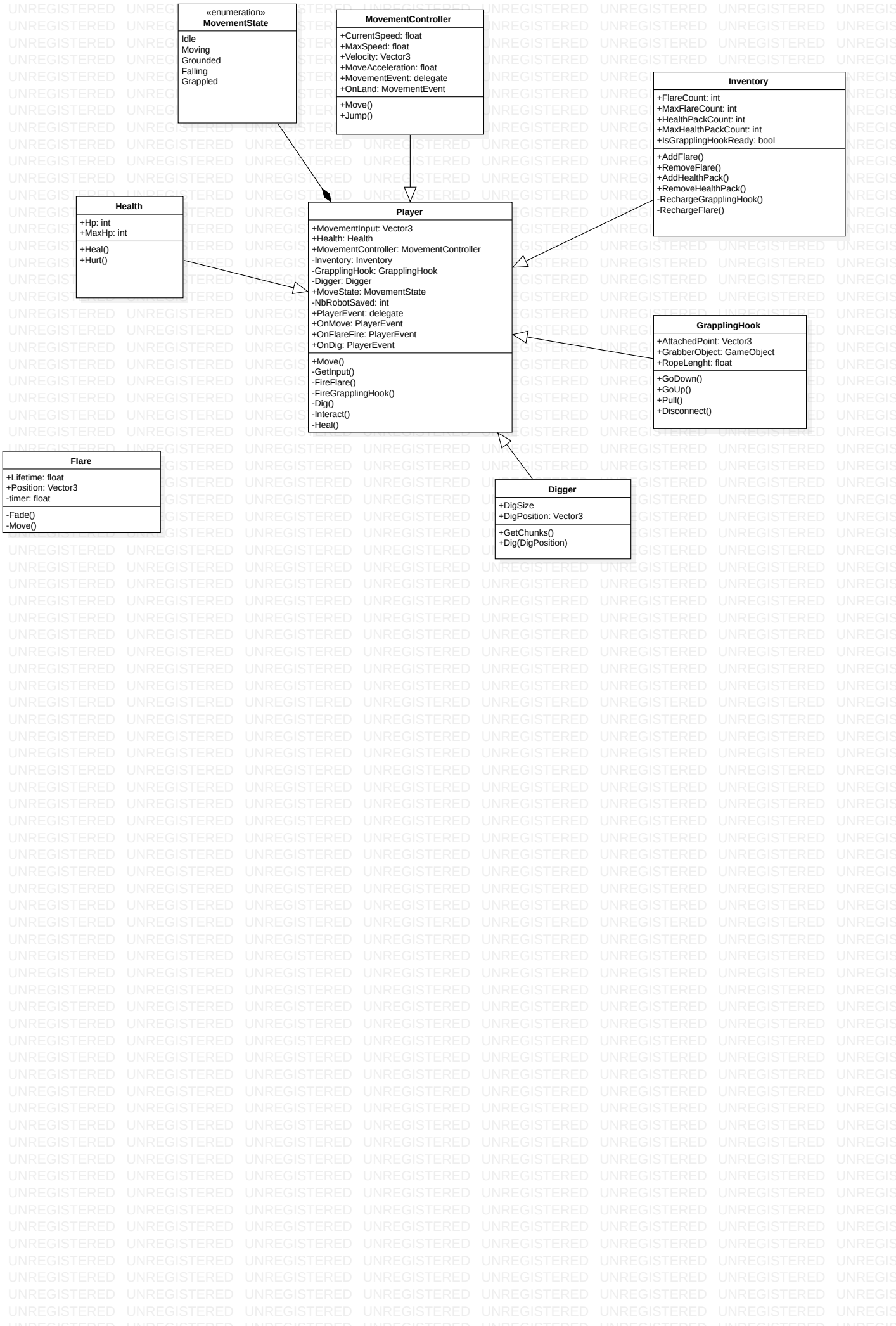
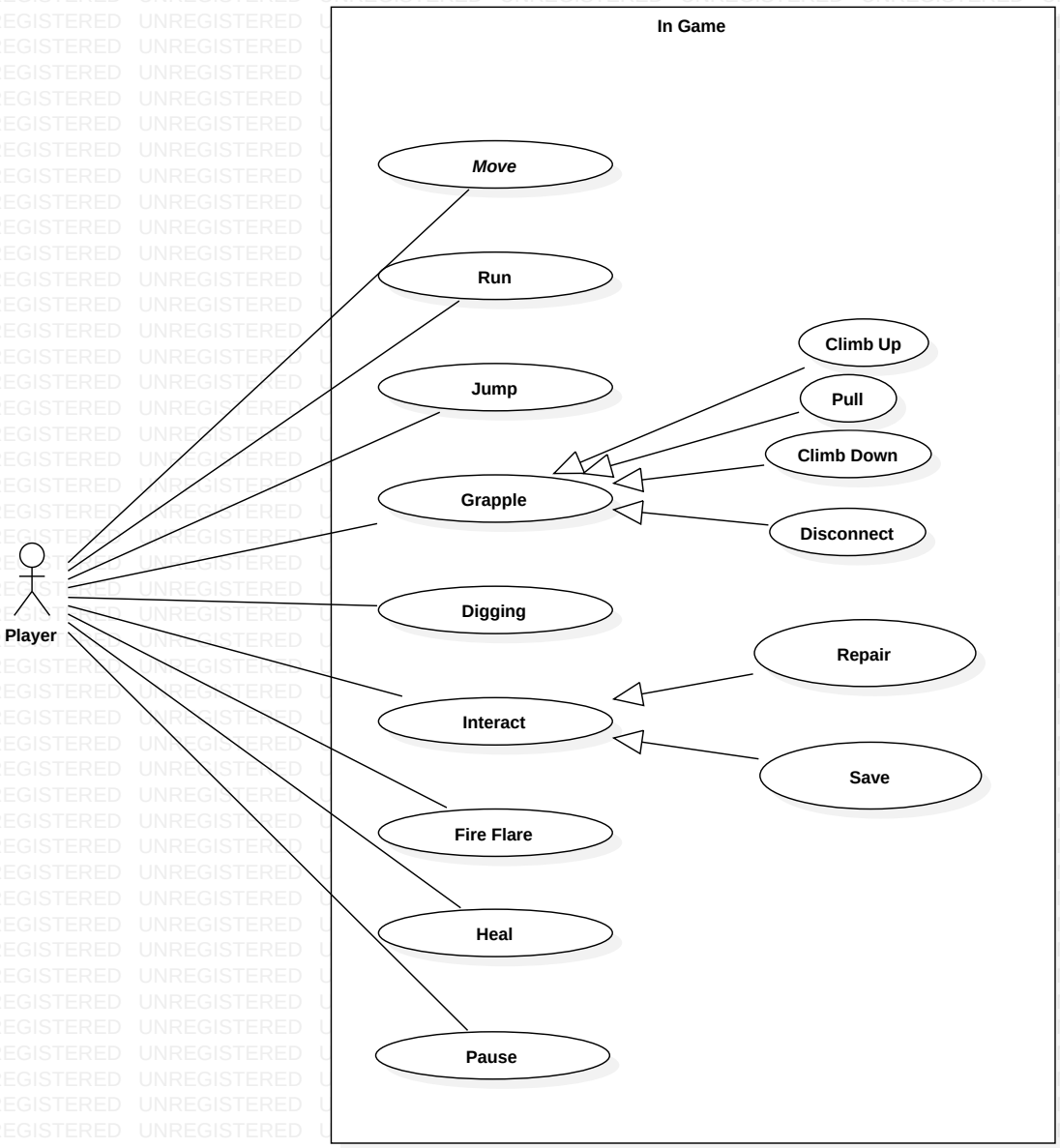
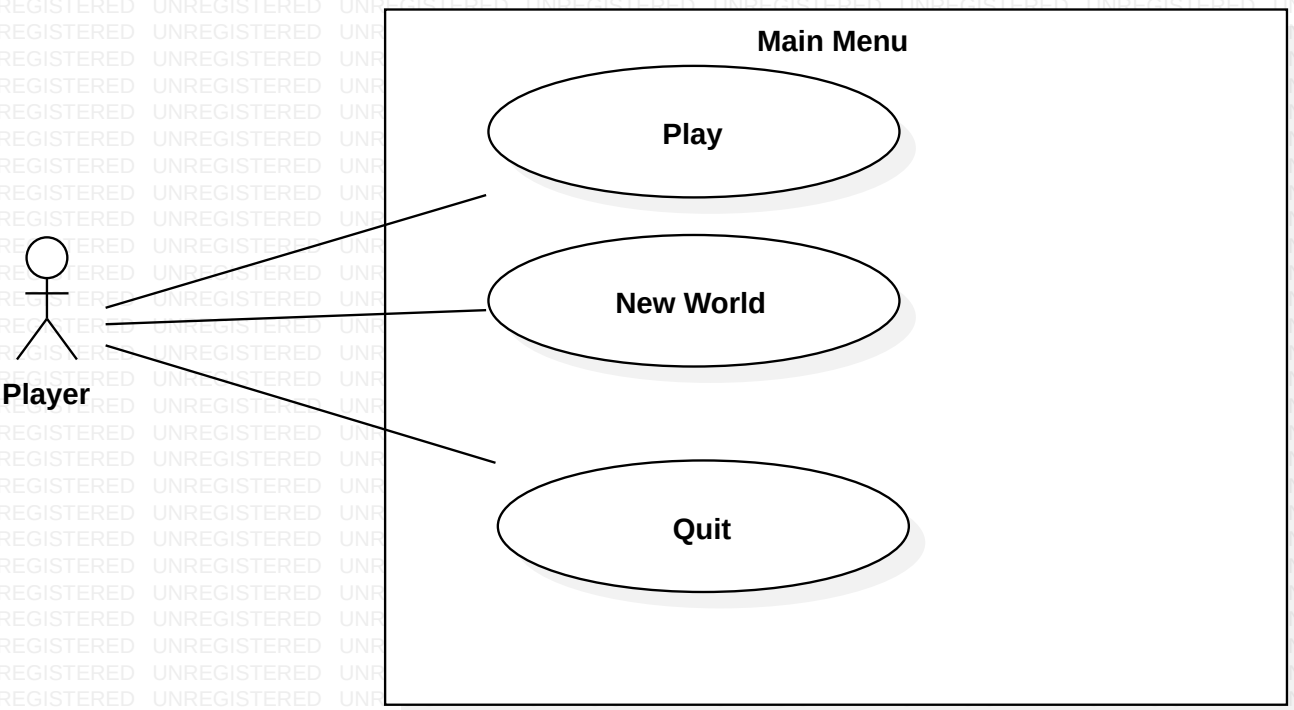
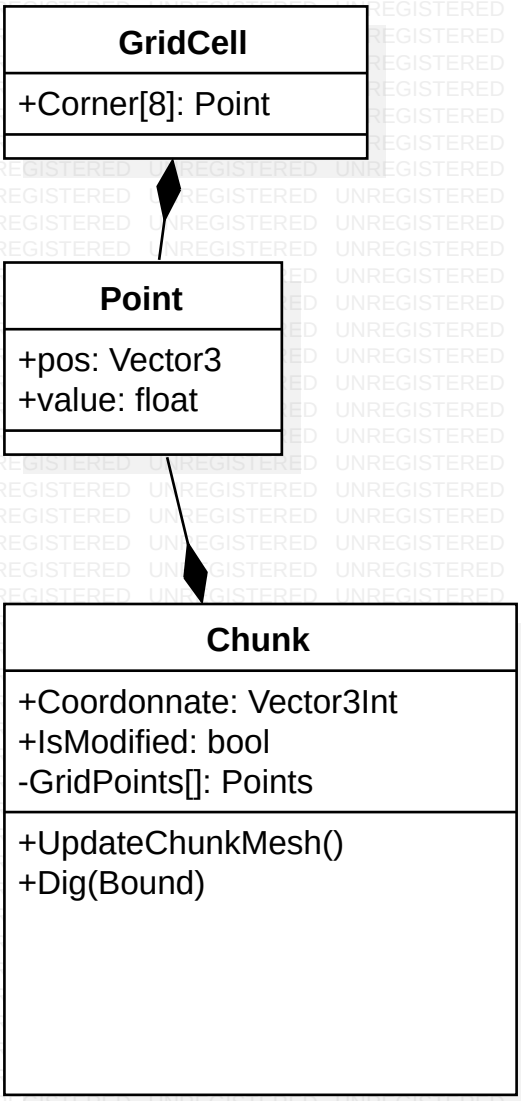

«enumeration»
MovementState
Idle
Moving
Grounded
Falling
Grappled
MovementController
+CurrentSpeed: float
+MaxSpeed: float
+Velocity: Vector3
+MoveAcceleration: float
+MovementEvent: delegate
+OnLand: MovementEvent
+Move()
+Jump()
Inventory
+FlareCount: int
+MaxFlareCount: int
+HealthPackCount: int
+MaxHealthPackCount: int
+IsGrapplingHookReady: bool
+AddFlare()
+RemoveFlare()
+AddHealthPack()
+RemoveHealthPack()
-RechargeGrapplingHook()
-RechargeFlare()
Health
+Hp: int
+MaxHp: int
+Heal()
+Hurt()
Player
+MovementInput: Vector3
+Health: Health
+MovementController: MovementController
-Inventory: Inventory
-GrapplingHook: GrapplingHook
-Digger: Digger
+MoveState: MovementState
-NbRobotSaved: int
+PlayerEvent: delegate
+OnMove: PlayerEvent
+OnFlareFire: PlayerEvent
+OnDig: PlayerEvent
+Move()
-GetInput()
-FireFlare()
-FireGrapplingHook()
-Dig()
-Interact()
-Heal()
GrapplingHook
+AttachedPoint: Vector3
+GrabberObject: GameObject
+RopeLenght: float
+GoDown()
+GoUp()
+Pull()
+Disconnect()
Flare
+Lifetime: float
+Position: Vector3
-timer: float
-Fade()
-Move()
Digger
+DigSize
+DigPosition: Vector3
+GetChunks()
+Dig(DigPosition)

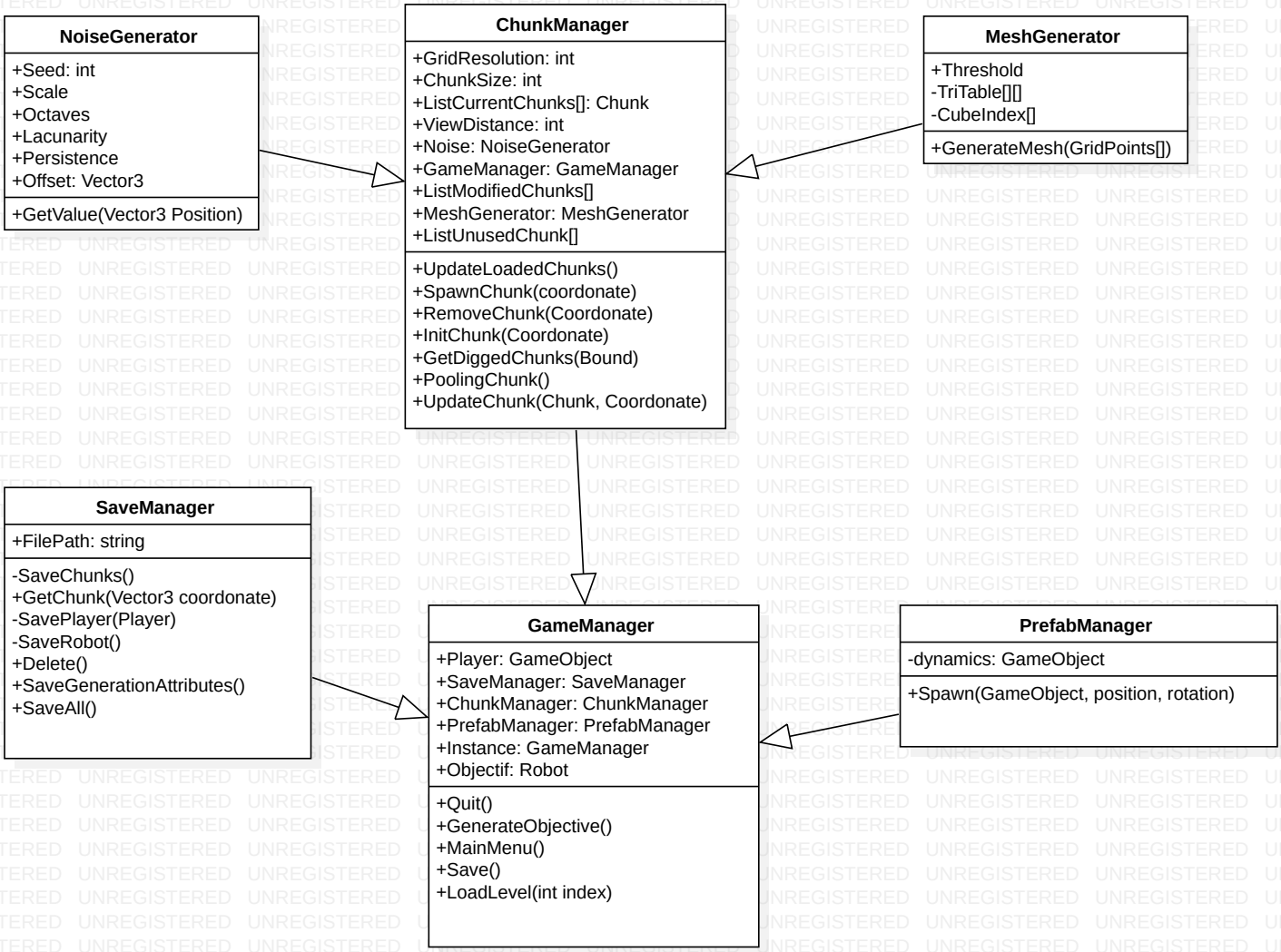
Model.UC_InGame

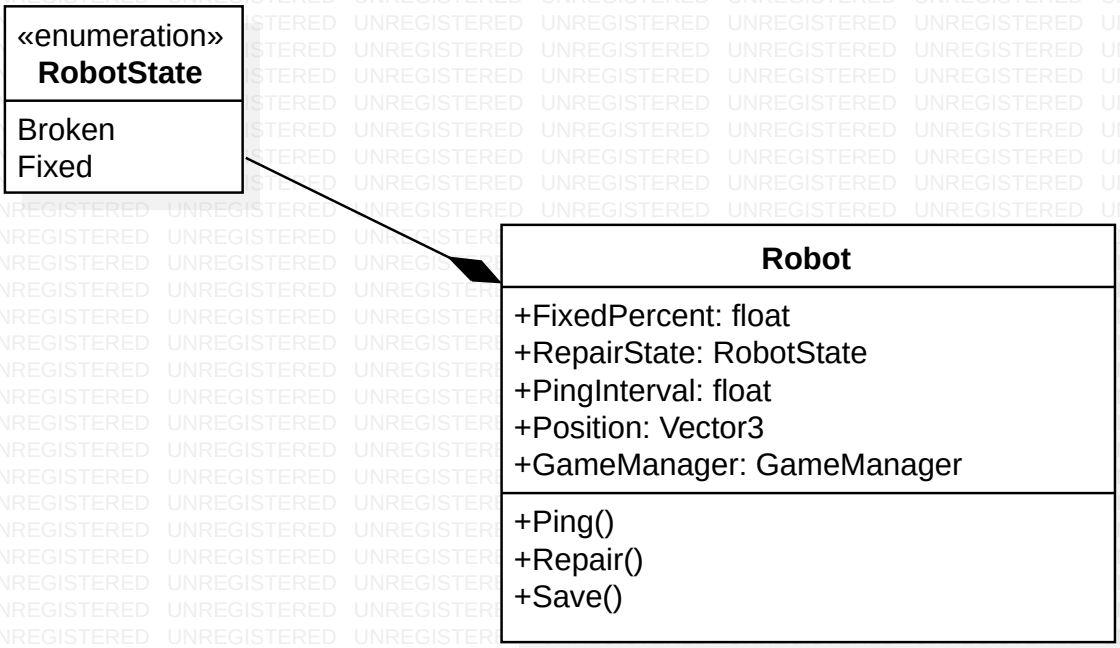


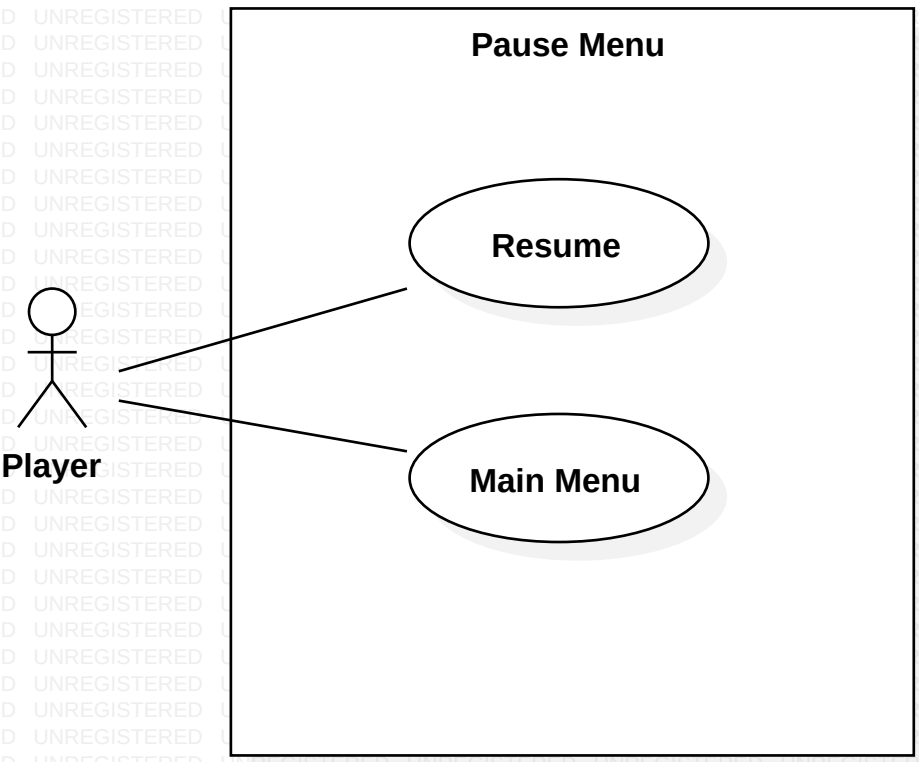
Model.UCD_MainMenu



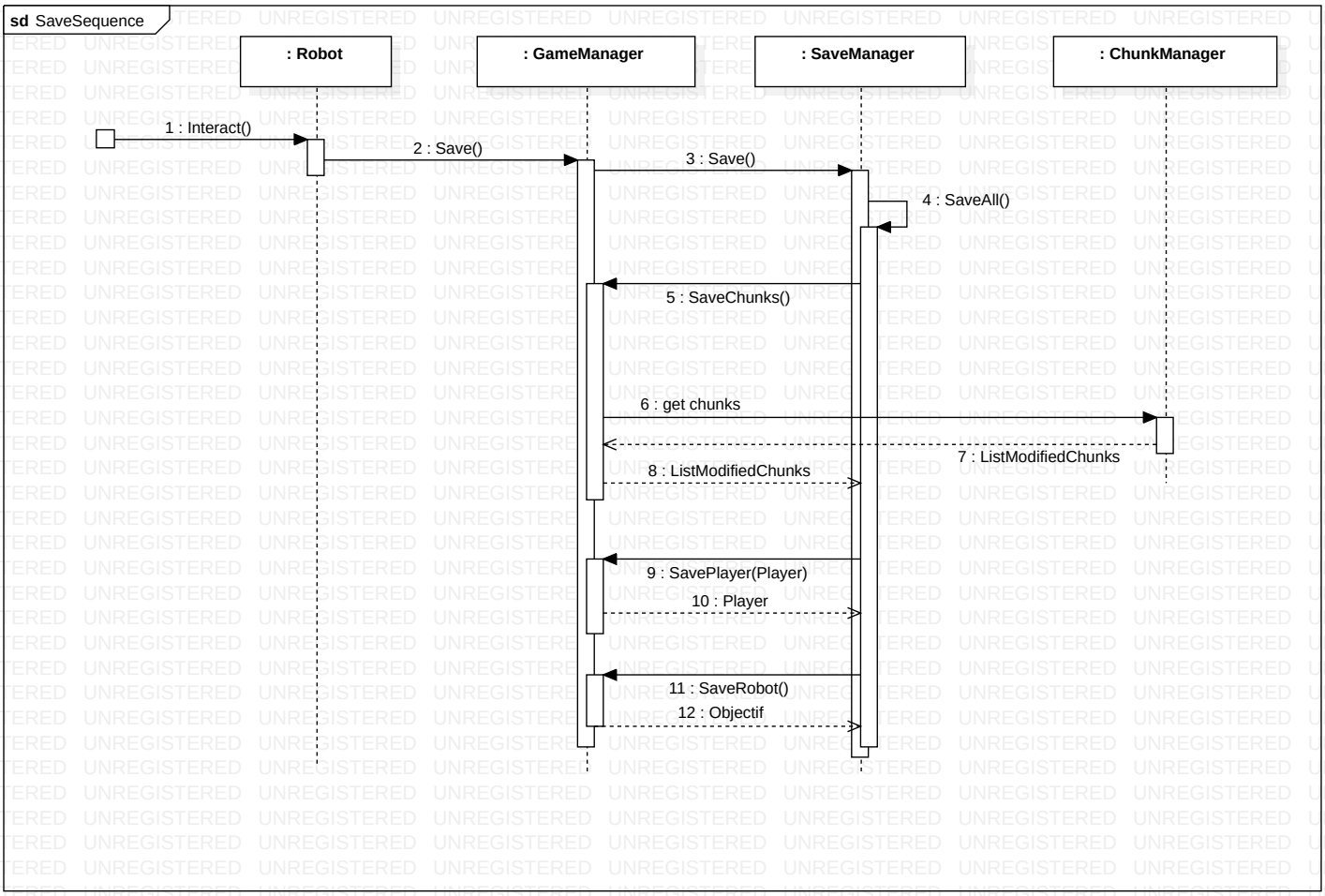

GridCell
+Corner[8]: Point
Point
+pos: Vector3
+value: float
Chunk
+Coordonnate: Vector3Int
+IsModified: bool
-GridPoints[]: Points
+UpdateChunkMesh()
+Dig(Bound)


NoiseGenerator
+Seed: int
+Scale
+Octaves
+Lacunarity
+Persistence
+Offset: Vector3
+GetValue(Vector3 Position)
ChunkManager
+GridResolution: int
+ChunkSize: int
+ListCurrentChunks[]: Chunk
+ViewDistance: int
+Noise: NoiseGenerator
+GameManager: GameManager
+ListModifiedChunks[]
+MeshGenerator: MeshGenerator
+ListUnusedChunk[]
+UpdateLoadedChunks()
+SpawnChunk(coordonate)
+RemoveChunk(Coordonate)
+InitChunk(Coordonate)
+GetDiggedChunks(Bound)
+PoolingChunk()
+UpdateChunk(Chunk, Coordonate)
MeshGenerator
+Threshold
-TriTable[][]
-CubeIndex[]
+GenerateMesh(GridPoints[])
SaveManager
+FilePath: string
-SaveChunks()
+GetChunk(Vector3 coordonate)
-SavePlayer(Player)
-SaveRobot()
+Delete()
+SaveGenerationAttributes()
+SaveAll()
GameManager
+Player: GameObject
+SaveManager: SaveManager
+ChunkManager: ChunkManager
+PrefabManager: PrefabManager
+Instance: GameManager
+Objectif: Robot
+Quit()
+GenerateObjective()
+MainMenu()
+Save()
+LoadLevel(int index)
PrefabManager
-dynamics: GameObject
+Spawn(GameObject, position, rotation)

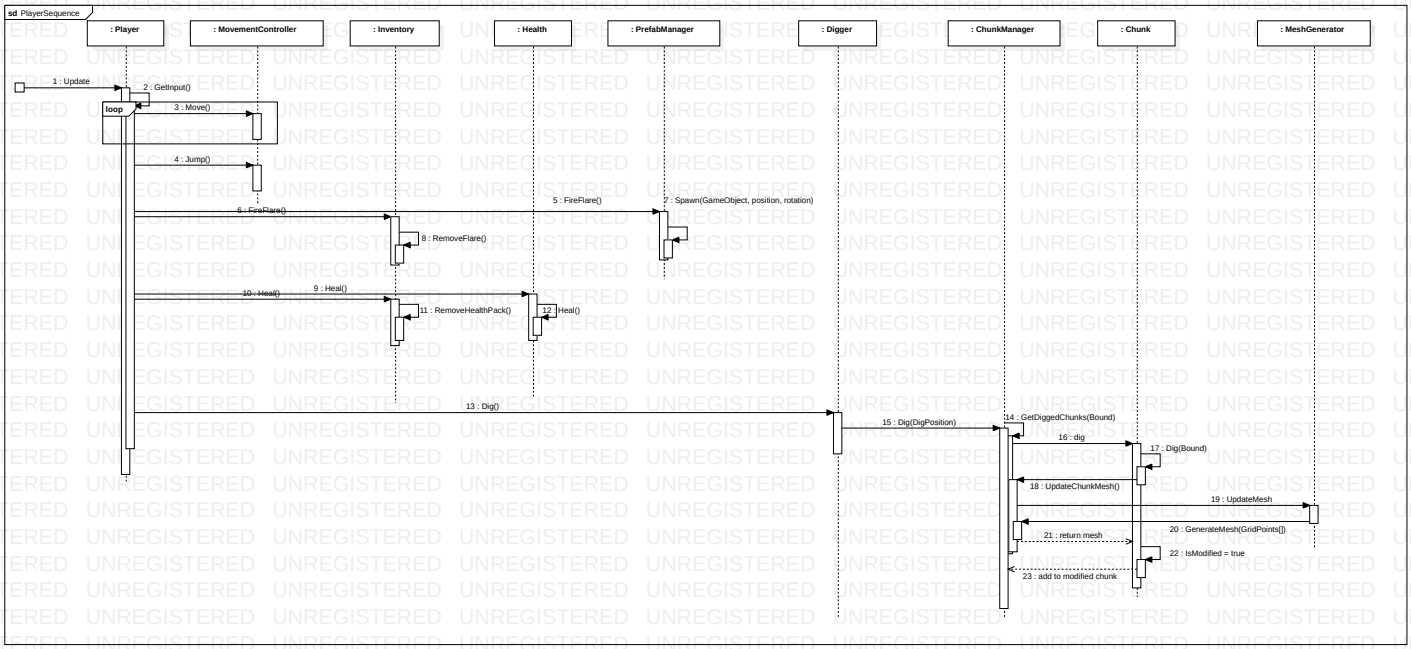

«enumeration»
RobotState
Broken
Fixed
Robot
+FixedPercent: float
+RepairState: RobotState
+PingInterval: float
+Position: Vector3
+GameManager: GameManager
+Ping()
+Repair()
+Save()



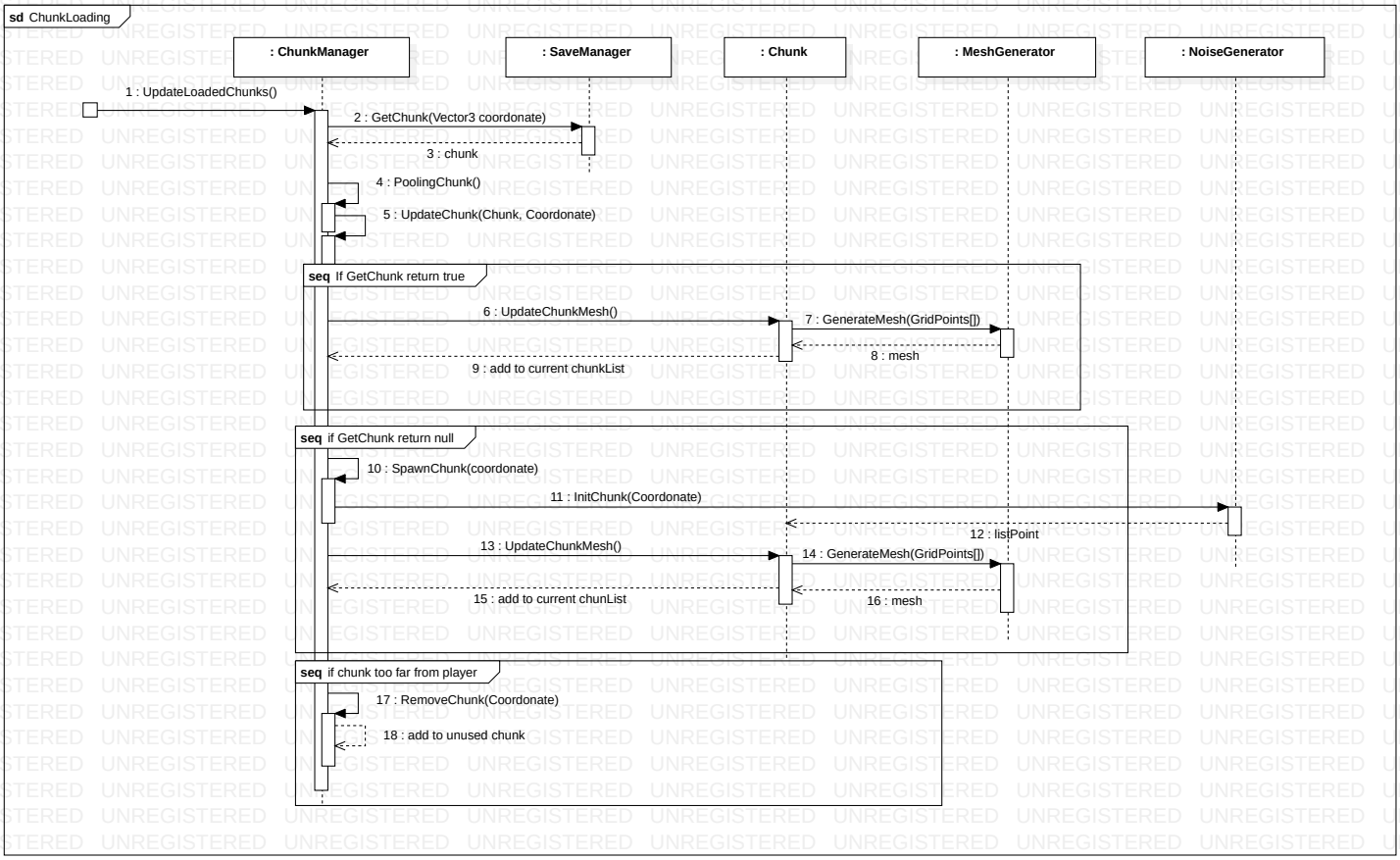
Model::Collaboration1::Interaction1::SaveSequence



Model Collaboration2 Interaction1 PlayerSequence



Collaboration1 : Interaction1 : ChunkLoading



Patron de conception

Object Pooling

Puisque le monde est infini, j'utilise des chunks (morceaux de terrain) qui seront charger autour du joueur. Pour ne pas instancier un nouveau chunk à proximité et détruire ceux qui sont trop loin, on réutiliser ceux qui devraient être détruit et on les met à jour pour le nouveau chunk qui devait être instancier.

Observer

Puisque plusieurs actions peuvent être effectuées, et plusieurs classes doivent être notifier, on va utiliser les delegates de Unity pour créer des évènements que n'importe quelle classe peut s'inscrire, et effectuer les actions nécessaires. Aussi, cela élimine plusieurs dépendances, donc si on décide d'effacer une classe, personne d'autre devrait en être pénaliser puisqu'il n'y pas de lien direct.

- (Dossier) Save
    - (fichier) Player.txt
    - (fichier) Robot.txt
    - (Dossier) Chunks
        - Chunk(0,0,0).txt
        - Chunk(0,0,1) .txt
        - Chunk(1,0,0) .txt
        - ….