Game Development
Kernmodule 3 AI

Mick Teunissen

Inhoud:

- Beschrijving Gameplay
- Beschrijving AI en implementatie
- Beschrijving gebruikte systeem
- Behaviour Tree (State-Diagram)
- UML
- PMI

Gameplay

Shinobi striker is een top down actie game waarbij de speler als ninja de samurai uit een dorp moeten zien te verslaan. Je krijgt hulp van een mede ninja die rookbommen kan gooien om de speler een opening kan geven om te vluchten.

Het hoofddoel is om alle samurai in het drop uit te schakelen. Dit kan de speler doen door shurikens naar de enemy samurai te gooien tot hp bar op 0 eindigt en de samurai is verslagen. Als alle 6 de samurai zijn uitgeschakeld heeft de speler gewonnen. Hebben de samurai er in geslaagd de hp van de speler tot 0 te brengen heeft de speler verloren.

AI en implementatie

Ik heb 2 soorten AI in de game verwerkt. De enemy samurai en de allied ninja. Om alle verschillende states en behaviours te managen heb ik Panda Behaviour Tree gebruikt. Ik heb hiervoor gekozen omdat deze redelijk makkelijk te begrijpen is en en best wat info over te vinden is online. Ik heb voor een bestaande behaviour tree gekozen om mezelf een hoop moeite te besparen omdat ik dan meer de focus op de game zelf kan leggen.

Ik moet wel eerlijk toegeven na lang testen en proberen het me niet is gelukt om de samurai te laten reageren op de smokebomb van de ally. Unity gaf een hoop Null reference errors en terwijl ik het duidelijk gedefinieerd had maargoed helaas werkte Unity niet mee en heb ik het na lang testen maar gelaten voor wat het is. De go back to patrol if in smoke functie van de enemy samurai er dus niet in verwerkt.

Ik heb wel een simpele game loop in de game zitten met een menu, win en lose scherm.

Gebruikte systeem

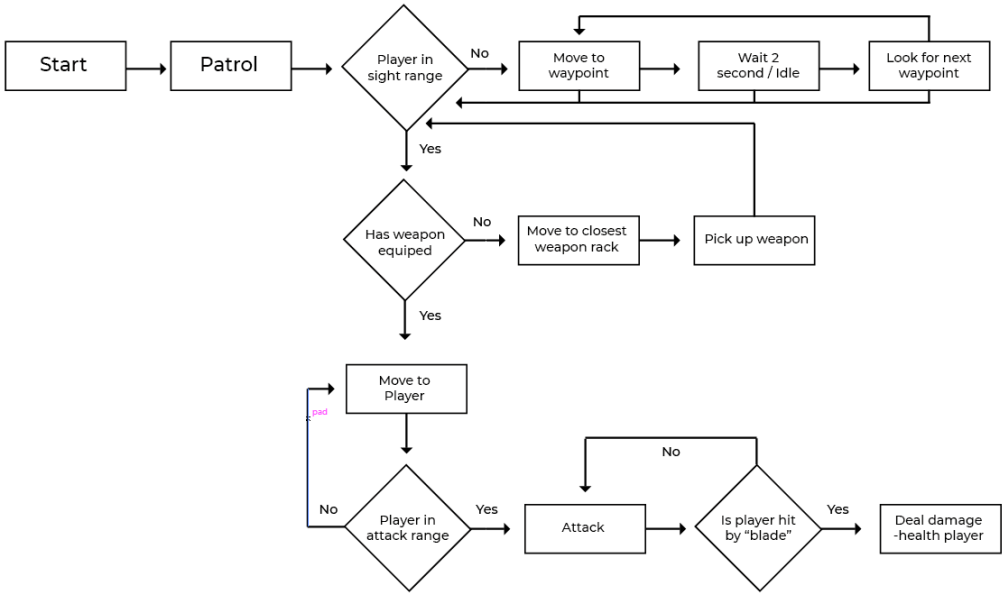
Voor het AI Pathfinding algoritme heb ik A* pathfinding gebruikt. Ik heb de zelfde code hiervoor gebruikt als die in opdracht 2. Na de lessen terug te hebben gekeken en wat tutorials op internet te hebben gevolgd van Sebastian Lague vond ik A* goed onder de knie te krijgen. Ik heb een aantal tutorials van Sebastian Lague letterlijk gevolgd om een begin te maken aan het A* algoritme. Daarna heb ik de code met eigen behaviours er aan toe te voegen goed kunnen uitbreiden.

De Samurai en de Ninja hebben allebei een ander behaviour script met verschillende states. De codes lijken best op elkaar maar ik heb toch voor 2 aparte scripts gekozen om het overzicht in het project te bewaren. In de behaviour scripts staan onder andere de Health, Movement, Actie, Patrol functies beschreven.

Voor de patrol functie van de samurai maak ik gebruik van een waypoint systeem zodat er consistente patrouille wordt aangehouden en dit de gameplay verbeterd in plaats van een bijvoorbeeld random position.

Behaviour Tree

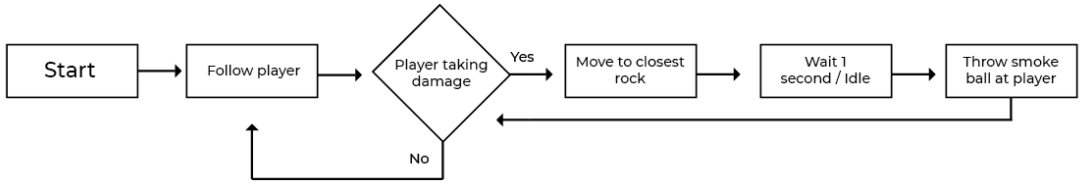
Samurai state diagram:



State	Purpose	Type of interaction
Patrol	Samurai patrols from waypoint to waypoint	world and AI
Grab weapon	Samurai goes to closest weapon rack	World, player and AI

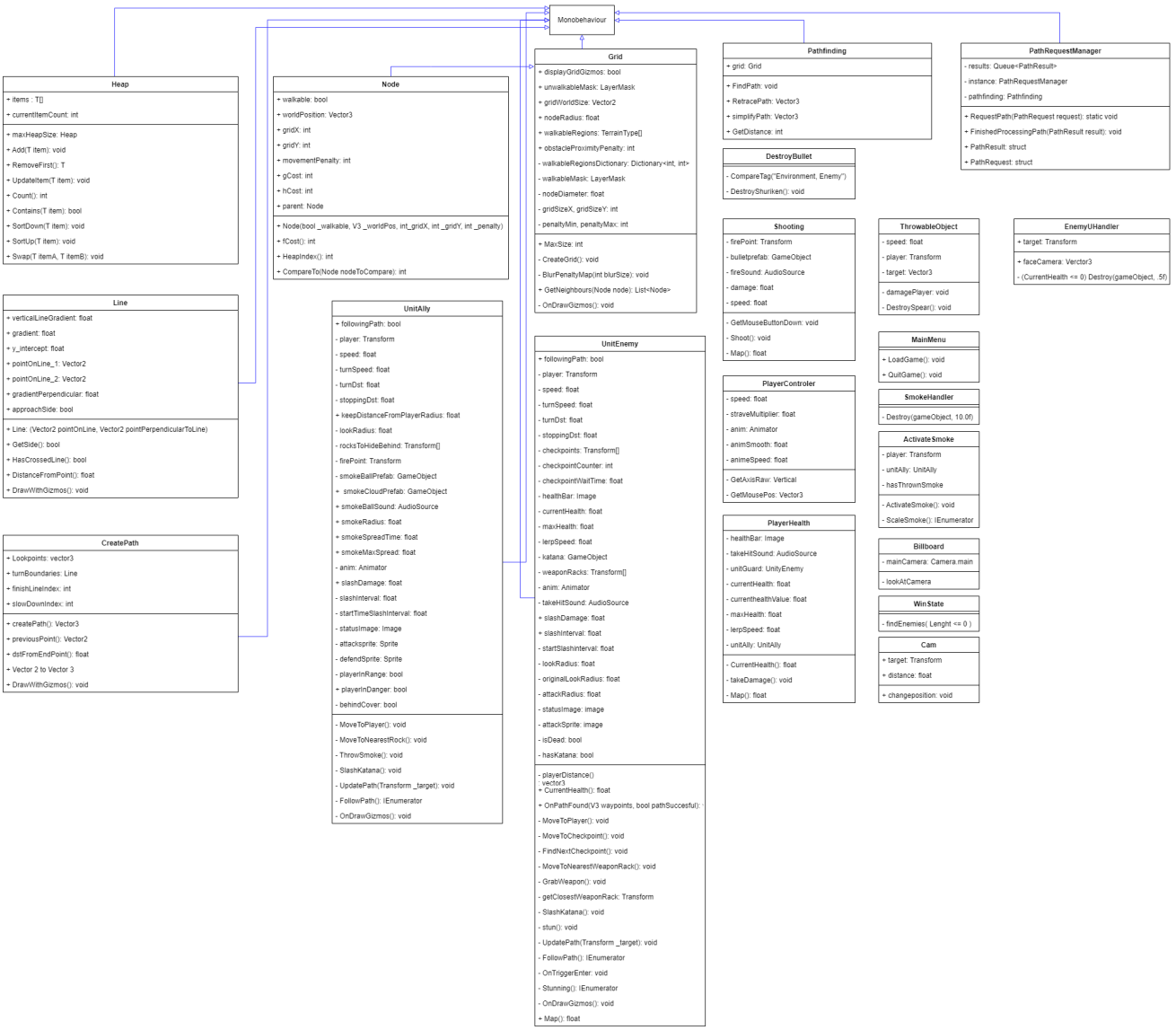
	when in sight of player and grabs weapon	
Follow player	Follows the player while in sightrange	Player and AI
Attack player	Attacks the player when in attack range	player and AI

Ninja state diagram:



State	**Purpose**	**Type of interaction**
Follow player	Follows the player around on range	Player and AI
Find closest cover / rock	Ninja finds cover to closest rock	World and AI
Protect Player / smoke	After in range of cover throws smoke at the player to distract the samurai	Player and AI

UML


Monobehaviour
Heap
+ items : T[]
+ currentItemCount: int
+ maxHeapSize: Heap
+ Add(T item): void
+ RemoveFirst(): T
+ UpdateItem(T item): void
+ Count(): int
+ Contains(T item): bool
+ SortDown(T item): void
+ SortUp(T item): void
+ Swap(T itemA, T itemB): void
Node
+ walkable: bool
+ worldPosition: Vector3
+ gridX: int
+ gridY: int
+ movementPenalty: int
+ gCost: int
+ hCost: int
+ parent: Node
+ Node(bool _walkable, V3 _worldPos, int_gridX, int _gridY, int _penalty)
+ fCost(): int
+ HeapIndex(): int
+ CompareTo(Node nodeToCompare): int
Grid
+ displayGridGizmos: bool
+ unwalkableMask: LayerMask
+ gridWorldSize: Vector2
+ nodeRadius: float
+ walkableRegions: TerrainType[]
+ obstacleProximityPenalty: int
- walkableRegionsDictionary: Dictionary<int, int>
- walkableMask: LayerMask
- nodeDiameter: float
- gridSizeX, gridSizeY: int
- penaltyMin, penaltyMax: int
+ MaxSize: int
- CreateGrid(): void
- BlurPenaltyMap(int blurSize): void
+ GetNeighbours(Node node): List<Node>
- OnDrawGizmos(): void
Pathfinding
+ grid: Grid
+ FindPath: void
+ RetracePath: Vector3
+ simplifyPath: Vector3
+ GetDistance: int
PathRequestManager
- results: Queue<PathResult>
- instance: PathRequestManager
- pathfinding: Pathfinding
+ RequestPath(PathRequest request): static void
+ FinishedProcessingPath(PathResult result): void
+ PathResult: struct
+ PathRequest: struct
Line
+ verticalLineGradient: float
+ gradient: float
+ y_intercept: float
+ pointOnLine_1: Vector2
+ pointOnLine_2: Vector2
+ gradientPerpendicular: float
+ approachSide: bool
+ Line: (Vector2 pointOnLine, Vector2 pointPerpendicularToLine)
+ GetSide(): bool
+ HasCrossedLine(): bool
+ DistanceFromPoint(): float
+ DrawWithGizmos(): void
UnitAlly
+ followingPath: bool
- player: Transform
- speed: float
- turnSpeed: float
- turnDst: float
- stoppingDst: float
+ keepDistanceFromPlayerRadius: float
- lookRadius: float
- rocksToHideBehind: Transform[]
- firePoint: Transform
- smokeBallPrefab: GameObject
+ smokeCloudPrefab: GameObject
+ smokeBallSound: AudioSource
+ smokeRadius: float
+ smokeSpreadTime: float
+ smokeMaxSpread: float
- anim: Animator
+ slashDamage: float
- slashInterval: float
- startTimeSlashInterval: float
- statusImage: Image
- attacksprite: Sprite
- defendSprite: Sprite
- playerInRange: bool
+ playerInDanger: bool
- behindCover: bool
- MoveToPlayer(): void
- MoveToNearestRock(): void
- ThrowSmoke(): void
- SlashKatana(): void
- UpdatePath(Transform _target): void
- FollowPath(): IEnumerator
- OnDrawGizmos(): void
UnitEnemy
+ followingPath: bool
- player: Transform
- speed: float
- turnSpeed: float
- turnDst: float
- stoppingDst: float
- checkpoints: Transform[]
- checkpointCounter: int
- checkpointWaitTime: float
- healthBar: Image
- currentHealth: float
- maxHealth: float
- lerpSpeed: float
- katana: GameObject
- weaponRacks: Transform[]
- anim: Animator
- takeHitSound: AudioSource
+ slashDamage: float
+ slashInterval: float
- startSlashInterval: float
- lookRadius: float
- originalLookRadius: float
- attackRadius: float
- statusImage: image
- attackSprite: image
- isDead: bool
- hasKatana: bool
- playerDistance()
vector3
+ CurrentHealth(): float
+ OnPathFound(V3 waypoints, bool pathSuccesful):
- MoveToPlayer(): void
- MoveToCheckpoint(): void
- FindNextCheckpoint(): void
- MoveToNearestWeaponRack(): void
- GrabWeapon(): void
- getClosestWeaponRack: Transform
- SlashKatana(): void
- stun(): void
- UpdatePath(Transform _target): void
- FollowPath(): IEnumerator
- OnTriggerEnter: void
- Stunning(): IEnumerator
- OnDrawGizmos(): void
+ Map(): float
CreatePath
+ Lookpoints: vector3
+ turnBoundaries: Line
+ finishLineIndex: int
+ slowDownIndex: int
+ createPath(): Vector3
+ previousPoint(): Vector2
+ dstFromEndPoint(): float
+ Vector 2 to Vector 3
+ DrawWithGizmos(): void
DestroyBullet
- CompareTag("Environment, Enemy")
- DestroyShuriken(): void
Shooting
- firePoint: Transform
- bulletprefab: GameObject
- fireSound: AudioSource
- damage: float
- speed: float
- GetMouseButtonDown: void
- Shoot(): void
- Map(): float
PlayerControler
- speed: float
- straveMultiplier: float
- anim: Animator
- animSmooth: float
- animeSpeed: float
- GetAxisRaw: Vertical
- GetMousePos: Vector3
PlayerHealth
- healthBar: Image
- takeHitSound: AudioSource
- unitGuard: UnityEnemy
- currentHealth: float
- currenthealthValue: float
- maxHealth: float
- lerpSpeed: float
- unitAlly: UnitAlly
- CurrentHealth(): float
- takeDamage(): void
- Map(): float
ThrowableObject
- speed: float
- player: Transform
- target: Vector3
- damagePlayer: void
- DestroySpear(): void
MainMenu
+ LoadGame(): void
+ QuitGame(): void
SmokeHandler
- Destroy(gameObject, 10.0f)
ActivateSmoke
- player: Transform
- unitAlly: UnitAlly
- hasThrownSmoke
- ActivateSmoke(): void
- ScaleSmoke(): IEnumerator
Billboard
- mainCamera: Camera.main
- lookAtCamera
WinState
- findEnemies(Lenght <= 0)
Cam
+ target: Transform
+ distance: float
+ changeposition: void
EnemyUHandler
+ target: Transform
+ faceCamera: Verctor3
- (CurrentHealth <= 0) Destroy(gameObject, .5f)

PMI

Voor en nadelen van Panda Behaviour tree en het A* algoritme

Plus

- De ai kan gemakkelijk en efficiënt de weg vinden met A*
- Panda behaviour tree is handig om mee in te komen omdat het een en ander al goed gedocumenteerd staat
- Je kan gemakkelijk alle states ordenen

Min

- Met te veel states kan het onoverzichtelijk worden
- Soms clashen states met elkaar en wordt debuggen een stukje lastiger

Interessant

- Soms lopen bepaalde states in elkaar vast. Bij mij kwam het voor dat de samurai naar de weapon rack wou lopen. En het vervolgens uit de range van de player liep en daarna weer terug ging naar zijn patrol route waar ik als de player weer stond. Hierdoor ging de samurai weer terug naar de weapon rack. Het was dus een eindeloze loop waar ik een oplossing voor moest bedenken.