3. ANALISA DAN DESAIN SISTEM

Dalam bab ini akan menjelaskan tentang desain dan alur sistem dari *game* ini. Pertama akan dijelaskan tentang penyesuaian menyeluruh dari desain yang digunakan di dalam *game.* Bagian selanjutnya akan membahas tentang desain sistem yang akan digunakan pada penelitian ini melalui *flowchart*. Bagian ketiga akan membahas tentang desain struktur AI yang akan digunakan. Pada bagian terakhir akan berisi desain *interface* yang akan digunakan pada *game.*

3.1 Analisa Masalah

Masalah utama yang diangkat pada skripsi ini adalah dimana NPC di dalam kebanyakan game sekarang tidak memiliki variasi perintah yang lebih, seperti kemampuan khusus atau pola serangan yang berbeda satu sama lain, serta alur permainan yang bersifat repetitif atau berulang sehingga tidak memaksa Player untuk memikirkan solusi dan yang pada akhirnya membuat permainan menjadi membosankan.

Untuk membuat NPC yang memiliki kemampuan bervariatif, digunakan metode *Behavior Tree* dikarenakan penggunaan fitur-fitur yang sangat membantu dalam mengembangkan AI yang kompleks namun masih mudah dibaca dan dipahami. Dalam *Behavior Tree*, penggunaan *Tasks* digunakan sebagai sebuah perintah yang harus dijalankan, *Tasks* bisa bervariasi seperti berjalan ke posisi tertentu, menyerang *Target*, atau melarikan diri ketika melihat *Target*. *Tasks* sendiri akan dibantu oleh *Decorator* untuk memberikan kondisi tambahan sebelum sebuah *Tasks* dijalankan, hal ini akan membuat NPC memiliki pola yang lebih bervariatif, seperti jika jumlah *Health* dari NPC dibawah rata-rata, maka NPC akan melarikan diri untuk mengisi *Health*-nya kembali. Juga penggunaan *Environment Query System* (EQS) yang berfungsi untuk mengumpulkan data dari dunia sekitar, sehingga membantu NPC dalam mengenali lingkungan sekitarnya dan membuat *Behavior Tree* menjadi lebih bervariatif, seperti mencari tempat persembunyian yang tidak bisa dilihat *Player*, atau melakukan serangan yang kuat jika *Player* terlalu dekat dengan NPC. Dalam game ini, akan dibuat 6 NPC dengan kemampuan yang bervariatif.

3.2 Game Desain

Pada sub bab ini, akan dijelaskan cara kerja dan detail dari *game* ini. Dalam *game* ini, tugas dari *Player* adalah bertahan hidup dan mengalahkan NPC dengan senjata yang ada dan harus menyelesaikan *stage* 3 untuk bisa menyelesaikan *game* ini.

3.2.1 *Game Storyline*

Aplikasi *Game* yang akan dibuat bernama “Dark Dream”. *Game* ini menceritakan tentang *Player* yang terjebak di dunia mimpi yang terus berulang dan tidak berakhir dan harus mencari sumber dari mimpi buruk ini untuk kabur dan Player selalu mengulangi mimpi buruk ini dengan memulai di tempat yang sama yaitu sebuah gereja tua. Semakin dalam *Player* mengeksplorasi mimpi ini, *Player* akan masuk ke dalam dunia mimpi yang bervariasi dengan masing-masing tantangan yang harus dilewati. *Player* juga akan menemukan makhluk-makhluk dengan pola dan perilaku yang berbeda. *Player* harus menyelesaikan teka-teki dan mencari petunjuk agar bisa melanjutkan eksplorasi. Pada akhirnya *Player* harus menghadapi sumber dari mimpi buruk ini untuk keluar.

3.2.2 *Component Game Design*

Ada beberapa *game component* yang dibutuhkan dalam pembuatan *game* ini. *Component-component* ini akan dibutuhkan agar dapat menyusun *game* ini. *Component* yang dimaksud adalah:

- *Stage*: Tempat di mana *Player* dan NPC berada. Tema dari *tutorial stage* adalah *gothic*, dan tema dari Stage 1, 2 dan 3 adalah perkotaan.
- *Stage Component*: Benda-benda yang terletak atau berada di dalam sebuah *Stage.* Benda-benda yang dimaksud adalah, pohon, bangunan, lampu, senjata, peluru senjata, dan *medkit* (obat).

3.2.3 *Player Design*

Di dalam *game* ini, *Player* dapat menggerakkan karakter di sekeliling daerah yang terbuka dan bermain dari segi *third-person perspective*. *Player* mempunyai wujud pria menggunakan jaket hitam dan celana Panjang seperti Gambar 3.1 dibawah ini.

Gambar 3.1 Tampilan *Player*

Player memiliki beberapa komponen penting yang membantu saat memainkan game, di antaranya adalah:

- *Health* : *Health* yang dimiliki oleh *Player* ditunjukkan dalam bentuk *bar*. *Health Bar* merupakan salah satu *component* yang sangat penting karena berfungsi sebagai indikator apakah *Player* masih dalam keadaan hidup, sekarat atau mati. Jumlah *Maximum Health* yang dimiliki oleh *Player* adalah 100. *Attack dari* NPC (*enemy*) yang diterima oleh Player akan memberikan *Damage Amount* yang akan mengurangi jumlah dari *Health* yang dimiliki oleh *Player*. Untuk memulihkan *Health Bar* diperlukan sebuah *medkit* yang bisa ditemukan disekitar daerah *stage* dan jumlah *Health* tidak bisa melebihi *Maximum Health.*

  $$Health = Health - Damage\ Amount \quad (1.1)$$

  Di mana:

  Damage Amount = Jumlah *Damage* yang di terima oleh *Player.*
- *Stamina* : *Stamina* juga merupakan salah satu component penting dalam membantu *Player* mempercepat proses eksplorasi di dalam game, *Stamina* yang dimiliki oleh *Player* di tunjukkan dalam bentuk *bar. Stamina Bar* berfungsi untuk *sprint* atau lari cepat. Dalam mode sprint, *Stamina Bar* akan otomatis berkurang dan ketika *Stamina* sudah habis, Player akan otomatis mengubah gerakan menjadi *jogging* atau lari santai dan *Stamina Bar* akan secara perlahan terisi kembali.
- *Ammo*: *Ammo* atau peluru merupakan *component* yang membuat Senjata bisa memberikan *damage* ke *enemy. Ammo* memiliki jenis sesuai dengan Senjata yang ada di dalam *game*, dan setiap *Ammo* memiliki kapasitasnya tersendiri. *Ammo* bisa ditemukan oleh *Player* di daerah sekitar *Stage.*
- Senjata : Senjata digunakan untuk mengalahkan musuh yang akan menyerang *Player,* senjata yang ada di *Game* adalah, *Pistol, Assault Rifle* dan *Sniper*, setiap senjata memiliki kapasitas peluru dan jenis *ammo* atau peluru yang berbeda, jika *Player* menemukan peluru dari senjata *Pistol* tetapi tidak memiliki senjata *Pistol,* maka peluru tersebut tidak akan bisa di ambil atau jika jumlah *Bag Ammo* yang ada di dalam *Pistol* masih penuh, peluru juga tidak akan bisa di ambil. Semua senjata memiliki *Damage* sebesar 10.
- *Inventory*: *Inventory* merupakan component sederhana yang digunakan sebagai tempat untuk menyimpan Senjata serta peluru yang ditemukan oleh Player dan juga *medkit,* tetapi *Player* hanya diperbolehkan membawa 3 *Medkit* dalam satu waktu, jika selama permainan menemukan *Medkit* tetapi dalam *Inventory* masih tersisa 3 *Medkit*, Player tidak akan bisa mengambilnya.
- *Medkit*: merupakan *component* yang bisa digunakan oleh *Player* ketika jumlah *Health* tidak sama dengan jumlah *Maximum Health*. *Medkit* akan menambah jumlah dari *Health* sebanyak 30, jika jumlah Health adalah 30 dan Player menggunakan *Medkit*, maka jumlah Health sekarang adalah

60 atau jika jumlah *Health* adalah 90 dan *Player* menggunakan *Medkit*, jumlah *Health* sekarang adalah 100 karena *Health* tidak boleh lebih dari *Maximum Health.*

3.2.4 *NPC (enemy) Design*

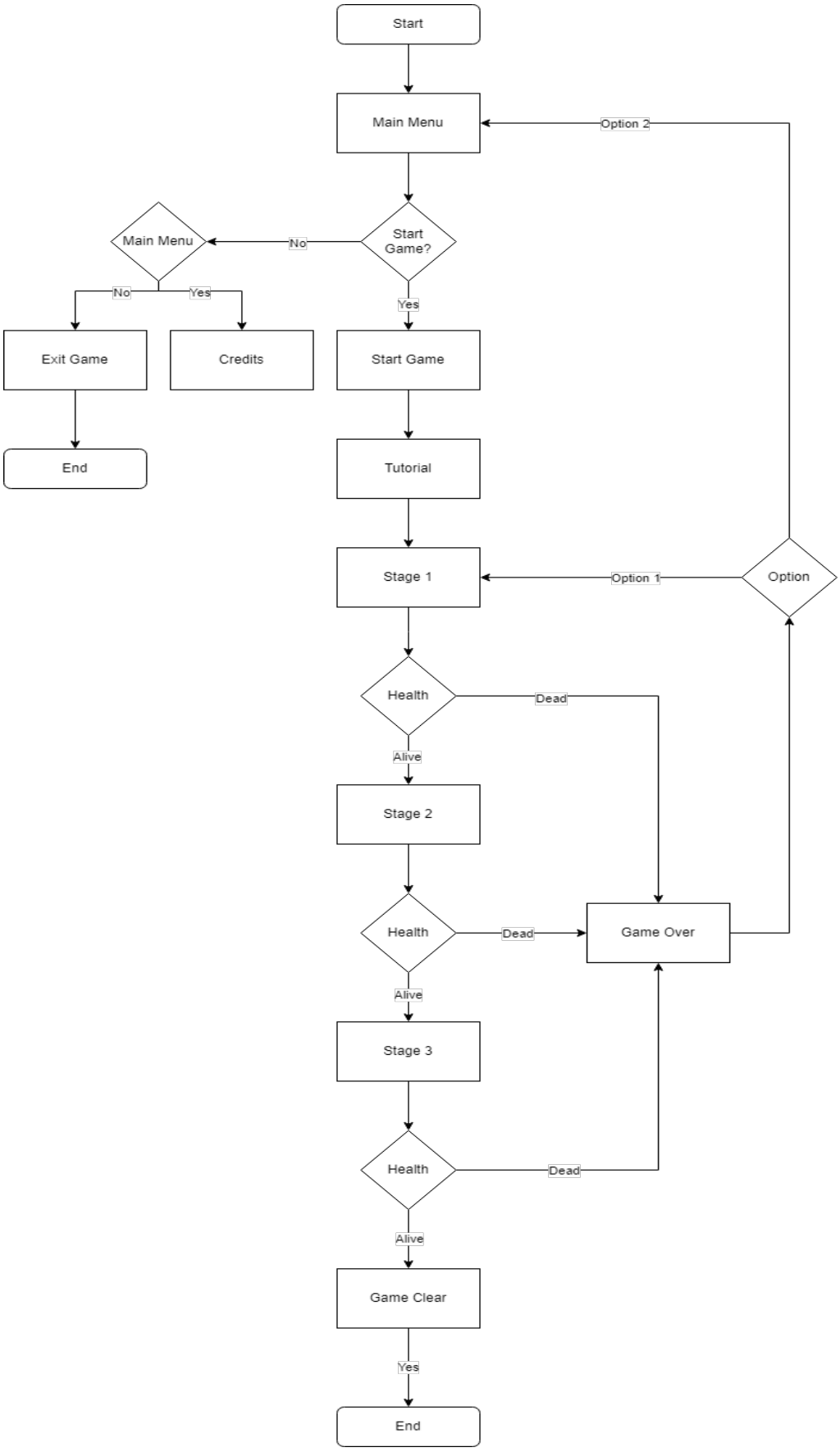
NPC di *game* ini ada 5 jenis yaitu, *Ghoul, Goblin, Skeleton, Zombie*, dan *Lich. Ghoul* dan *Goblin* memiliki variasi pergerakan dan kemampuan yang sama, *Zombie* memiliki kemampuan tambahan untuk berpatroli atau *patrol* sedangkan *Skeleton* memiliki pergerakan dan kemampuan yang sama, tetapi kemampuan tambahan yaitu *blocking. Lich* tidak bisa menggunakan kemampuan *blocking* tetapi, memiliki kemampuan yang lebih bervariasi seperti teleportasi dan *restore*. Setiap NPC memiliki beberapa *component,* di antaranya adalah:

- *Attack*: *Attack* adalah salah satu kunci component penting yang dimiliki oleh *enemy*. Component ini berguna untuk memberikan *damage* atau mengurangi jumlah *Health* yang dimiliki oleh *Player*. Setelah *enemy* menjalankan *Attack,* akan ada cooldown atau jeda waktu sebelum *enemy* bisa menjalankan kembali fungsi *Attack.* Setiap NPC (*enemy*) memiliki jumlah *Damage Amount* yang berbeda:
  1. *Ghoul* : 10 *Damage*
  2. *Goblin* : 5 *Damage*
  3. *Zombie* : 10 *Damage*
  4. *Skeleton*: 15 *Damage*
  5. *Lich* : 10 *Damage*
- *Health: Health* sama seperti *Player* sebagai indikator apakah *enemy* dalam keadaan hidup atau mati dan ditunjukkan dalam bentuk *bar*. *Health Bar* akan berkurang jika menerima *damage* dari Senjata *Player.* Setiap NPC (*enemy*) memiliki jumlah *Health* yang berbeda:
  1. *Ghoul* : 80 *Health*
  2. *Goblin* : 60 *Health*
  3. *Zombie* : 250 *Health*
  4. *Skeleton*: 150 *Health*
  5. *Lich* : 350 *Health*
- *Chase*: *Chase* adalah salah satu component yang sangat penting yang dimiliki oleh *enemy.* Fungsi dari *component* ini adalah mengejar *Player, Chase* akan berjalan ketika *Player* sudah memasuki radius penglihatan *enemy* atau ketika *Player* mengurangi *Health* yang dimiliki oleh *enemy* dengan menembaknya.
- *Investigating*: *Component Investigating* akan berjalan ketika *enemy* kehilangan posisi atau pandangan dari *Player, enemy* akan mulai mencari atau *investigate* keberadaan *Player* dalam

radius dari posisi terakhir *Player* terlihat, jika gagal menemukan keberadaan *Player* dalam jangka waktu yang ditentukan, *enemy* akan kembali ke *component* berikutnya yaitu *Patrol.*

- *Patrol*: *Component Patrol* adalah fungsi yang akan langsung dijalankan ketika *game* dimulai, *component* ini memberikan kemampuan kepada *enemy* untuk berjalan sesuai titik-titik yang sudah ditentukan, jika *enemy* sudah sampai di titik terakhir dari *Patrol, enemy* akan berbalik dan berjalan menuju titik awal dari *Patrol. Component* ini akan berjalan terus-menerus sampai *enemy* mendapatkan pandangan dari *Player* atau menerima *damage* dari *Player.*
- *Freeze:* Ketika *enemy* menerima *damage* dari *Player,* pergerakan dari *enemy* akan berhenti sejenak dan akan lanjut *Chase* atau *Attack Player* kembali. Tetapi, jika *enemy* sedang menjalankan *Attack*, *Freeze* tidak akan dijalankan.
- *Block: Component Block* akan berjalan ketika *enemy Skeleton Sword* mendeteksi jika Player sedang melakukan serangan, dan jika serangan mengenai *enemy Skeleton,* serangan tersebut akan terblokir dan tidak mengurangi jumlah *Health* dari *enemy Skeleton. Component Block* memiliki *cooldown* atau jeda waktu selama 2 detik sebelum bisa digunakan lagi supaya *enemy* tidak terus-menerus memblokir serangan dari *Player.* Ketika *Component Block* sedang dalam *cooldown, enemy Skeleton* akan menerima *damage* dan menjalankan *Freeze.*
- *Restore*: *Component Restore* akan berjalan ketika jumlah *Health* dari *enemy Lich* dibawah 20%, jika jumlah *Health* sudah memenuhi ketentuan, *enemy Lich* akan lari dari *Player* dan berusaha mencari tempat yang sedang tidak bisa dilihat oleh *Player* untuk bersembunyi dan memulihkan *Health* sebanyak 5% dalam waktu 1 detik. Setelah waktu *Restore* berakhir, *enemy Lich* akan keluar dari persembunyian dan kembali menyerang *Player.*
- *Teleport*: *Component Teleport* digunakan oleh *enemy Lich* untuk berpindah dari satu titik ke titik berikutnya dengan sangat cepat sehingga Player harus selalu memeriksa lingkungan sekitar untuk menebak lokasi yang *enemy Lich* tuju. Setelah *teleport, enemy Lich* akan melakukan serangan yang langsung dilanjutkan dengan menjalankan teleport kembali, jika variasi serangan yang dimiliki oleh *Lich* dalam keadaan *Cooldown* atau dalam jeda waktu, maka akan menjalankan *component strafe* untuk mengelilingi *Player.*
- *Strafe*: *Component Strafe* memungkinkan *enemy* untuk mengepung *Player* dengan berjalan mengelilingi *Player* dalam radius yang ditentukan. Ketika sudah di dekat *Player*, *enemy* tidak akan langsung menjalankan *Component Attack* tetapi akan menjalankan *Strafe* untuk durasi yang sudah ditentukan baru kemudian menjalankan *Attack.* Jika *Attack* selesai, *enemy* akan menjalankan *Strafe* dan proses akan diulangi terus-menerus.

3.2.5 *Gameplay Desain*

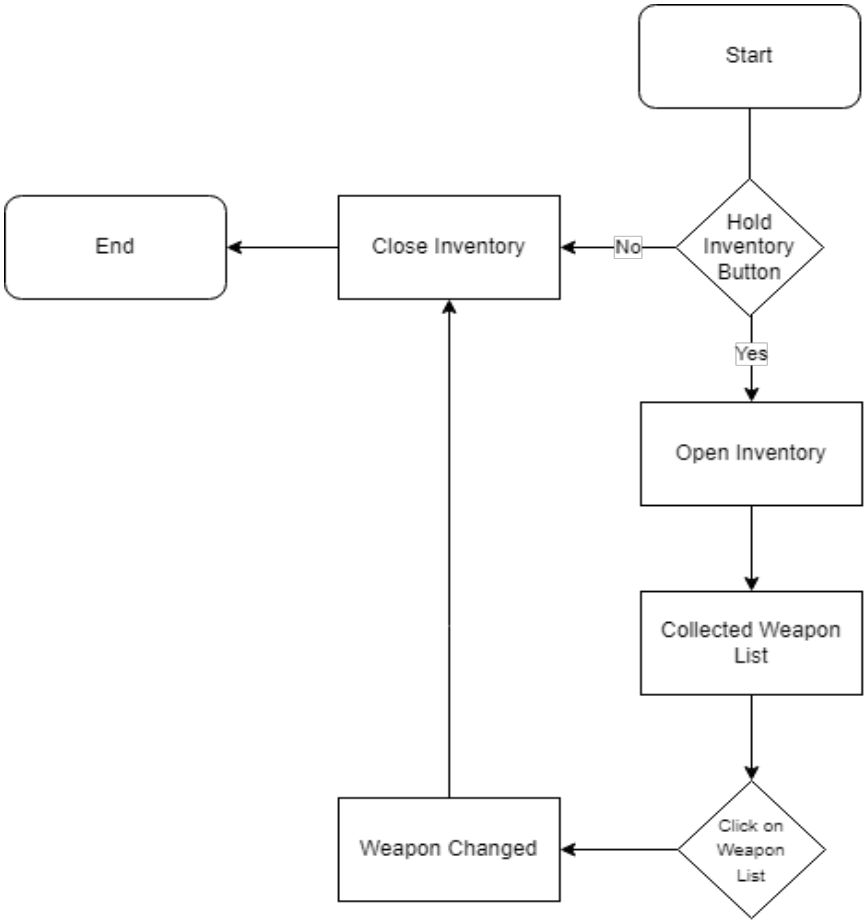


Gambar 3.2. *Flowchart* Proses *Gameplay*

Game akan dimulai seperti tampilan *Flowchart* Gambar 3.2. Keterangan yang bisa diambil adalah:

- *Player* akan memulai permainan dari *Main Menu* yang memiliki 3 pilihan, yaitu *Start Game*, *Credits* dan *Exit Game.*
- Menu *Start Game, Player* akan memasuki *tutorial stage* untuk mendapat informasi cara permainan dari *Game* ini, *Menu Credits* akan menunjukkan nama dari pembuat *Game* yaitu nama dari Penulis, menu *Exit Game* akan mengeluarkan *Player* dari *Game.*
- Setelah menyelesaikan *tutorial stage, Player* bisa menuju ke *Stage 1* dan mencoba untuk menyelesaikannya, tetapi jika *Health* dari *Player* sudah mencapai 0 atau *dead*, permainan otomatis berhenti dan masuk ke menu *Game Over. Game Over* memiliki 2 pilihan, memulai kembali dari *tutorial stage* atau ke *Main Menu.*
- Jika masih hidup, *Player* bisa melanjutkan permainan ke *Stage 2*, jika di *Stage 2 Player* bisa menyelesaikan permainan dan masih hidup, bisa melanjutkan permainan ke *Stage 3,* jika tidak, akan kembali masuk ke menu *Game Over*. Jika *Player* bisa menyelesaikan permainan dan masih hidup di *Stage 3*, *game* akan selesai, jika tidak, kembali lagi masuk ke menu *Game Over.*

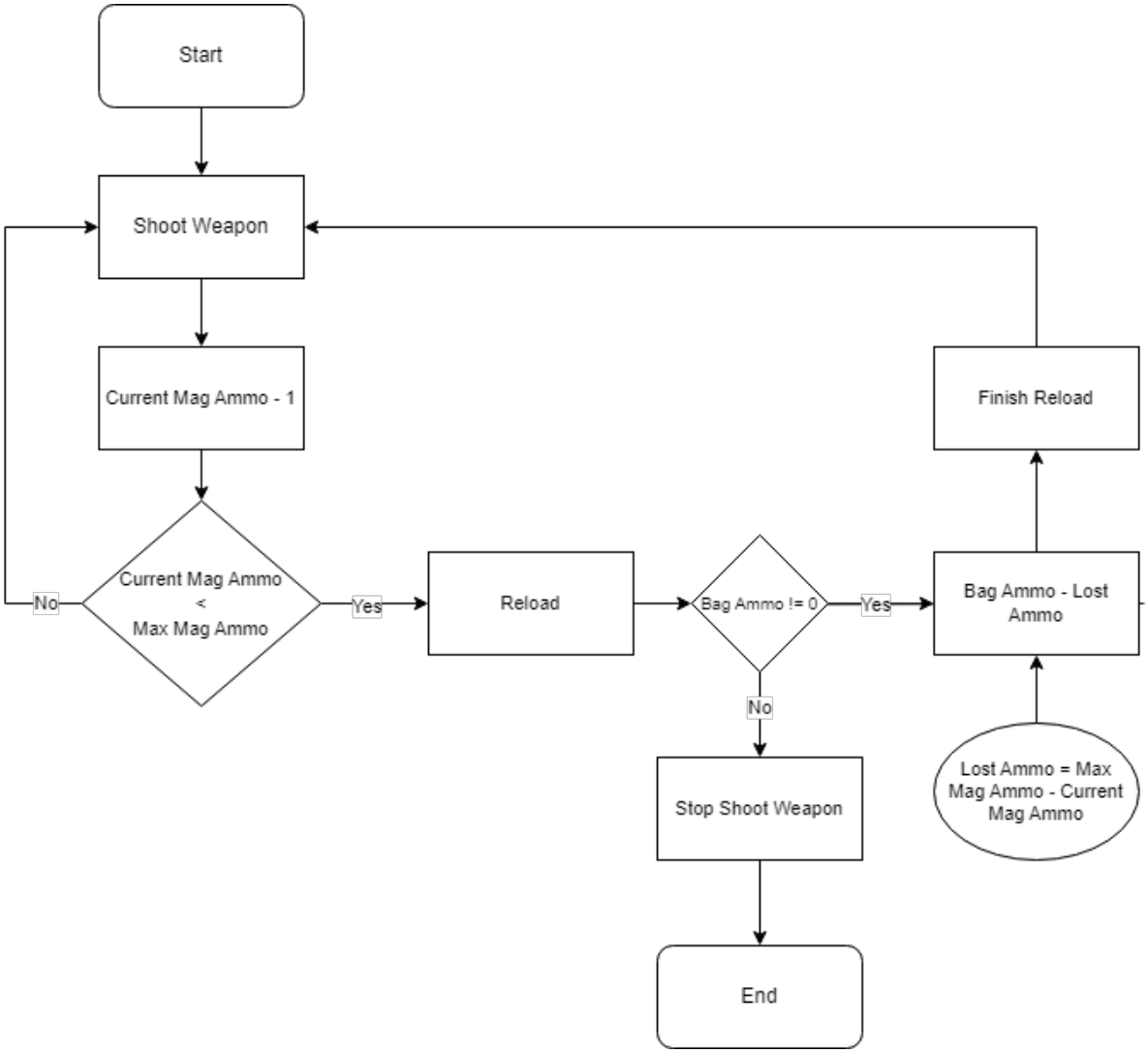
3.2.5.1 Proses *Inventory*



Gambar 3.3 *Flowchart* proses *Inventory*

Proses dari *Inventory* dijalankan ketika Player ingin mengganti senjata yang telah diperolah selama bereksplorasi. *Player* harus menahan tombol untuk membuka layar *Inventory,* jika tombol dilepas layar Inventory akan langsung tertutup, setelah Player memilih senjata pengganti, inventory juga akan langsung tertutup.

3.2.5.2 Proses *Shoot Weapon* dan *Reload*

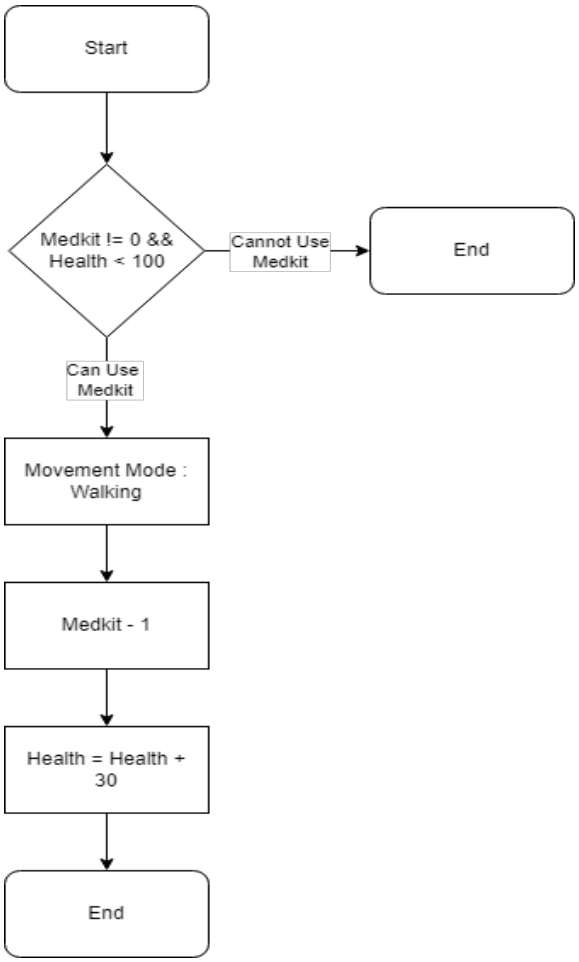


Gambar 3.4 *Flowchart* Proses *Shoot Weapon* dan *Reload*

Setiap Player menembak senjata, *Current Mag Ammo* berkurang 1. Kapasitas Senjata dibagi menjadi dua, yaitu *Mag Ammo dan Bag Ammo, Mag Ammo* adalah jumlah peluru yang sedang berada di dalam sebuah senjata dan bisa langsung digunakan, sedangkan *Bag Ammo* adalah jumlah peluru yang bisa *Player* kumpulkan ketika sedang bereksplorasi, *Player* tidak akan bisa menyerang *enemy* jika *Mag Ammo* dan *Bag Ammo* berjumlah 0. Misal, *Pistol* memiliki maksimum *Mag Ammo* sebanyak 20 dan maksimum *Bag Ammo* sebanyak 60, *Player* menembak *enemy* sebanyak 15 kali, 15 peluru ini

adalah *Lost Ammo*. *Player* bisa mengisi *Current Mag Ammo* ke jumlah maksimum 20 dengan mengurangi *Bag Ammo* sebanyak jumlah dari *Lost Ammo*, sisa *Bag Ammo* adalah 45 dan untuk mengisi *Bag Ammo* ke jumlah maksimum 60, Player harus mencarinya dengan bereksplorasi. Jika Player mencoba menembak senjata dengan *Mag Ammo* yang kosong, tetapi masih memiliki *Bag Ammo,* senjata akan mengisi *Mag Ammo* secara otomatis.

3.2.5.3 Proses *Medkit*



Gambar 3.5 Proses penggunaan *Medkit*

Ketika *Player* ingin menggunakan Medkit, akan dilakukan pengecekan dahulu apakah *Player* memiliki Medkit atau tidak dan apakah jumlah *Health* dibawah 100. Ketika kedua kondisi sudah terpenuhi, mode pergerakan *Player* akan di ubah menjadi berjalan, jumlah *Medkit* di kurangi 1 dan jumlah *Health* ditambah sebanyak 30. Jika tidak, maka *Player* tidak bisa menggunakan Medkit.

3.3 *Artificial Intelligence* (AI) Desain

Pada sub bab ini, akan dijelaskan struktur AI yang akan digunakan. Dalam pembuatan game ini, AI yang digunakan dalam mengontrol pergerakan NPC enemy adalah *Behaviour Tree*. *Behaviour Tree* digunakan sebagai AI yang mengontrol cara perilaku pergerakan dan juga pengelihatan yang

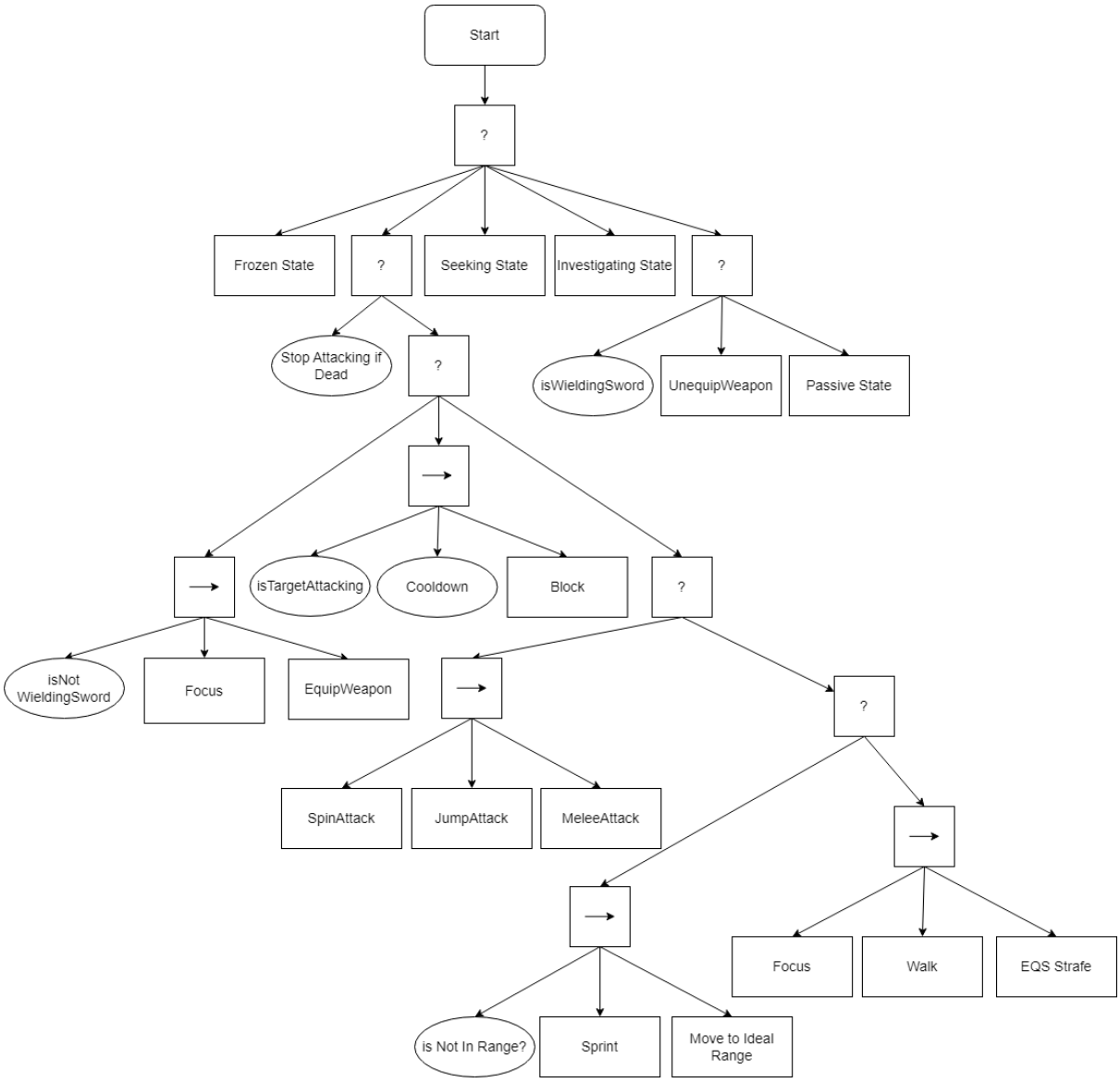
bertujuan untuk mengejar dan juga membunuh player. Selama player masih belum menyelesaikan game, maka Behaviour Tree dari NPC akan tetap berjalan hingga player menyelesaikan game atau mati (*Dead*).

3.3.1 *Behavior Tree*

Behavior Tree adalah salah satu metode yang sering digunakan dalam pembuatan AI yang ada dalam *game* dan akan diimplementasikan kepada AI NPC. Proses pembentukan Behavior Tree akan menggunakan beberapa elemen yang digunakan sebagai fondasi untuk mengatur AI NPC yaitu, *Tasks*, *Decorator*, *Services*, dan *Environment Query System* (EQS). Proses Behavior Tree akan tetap berjalan selama NPC ada di dalam *game,* karena proses *Behaviour Tree* akan mengulang kembali dari awal pada saat mencapai akar dari *Tree* tersebut, dan akan berpindah ke cabang yang lain jika kondisi dari cabang tersebut terpenuhi terlebih dahulu atau arah cabang tersebut berada pada bagian terkiri pada *Behaviour Tree*. Proses jalan *Behaviour Tree* akan berjalan dari cabang terkiri hingga cabang terkanan dari sebuah *tree.* Selama kondisi sebuah cabang masih dapat terpenuhi, ada kesempatan di mana sebuah *task* akan dilakukan berulang kali di karena kan kondisi dari cabang tersebut masih terpenuhi. Setelah kondisi dari sebuah cabang tidak lagi terpenuhi, maka akan berpindah ke cabang lain yang berada di sebelah kanan cabang sebelumnya hingga mencapai cabang terkanan dari sebuah *tree*. Jika kondisi dari cabang terkanan sudah tidak bisa dipenuhi lagi maka akan berpindah lagi ke cabang terkiri dari sebuah *tree.*

3.3.2 *Behavior Tree Design*

3.3.2.1 NPC *Skeleton* (*Sword*)

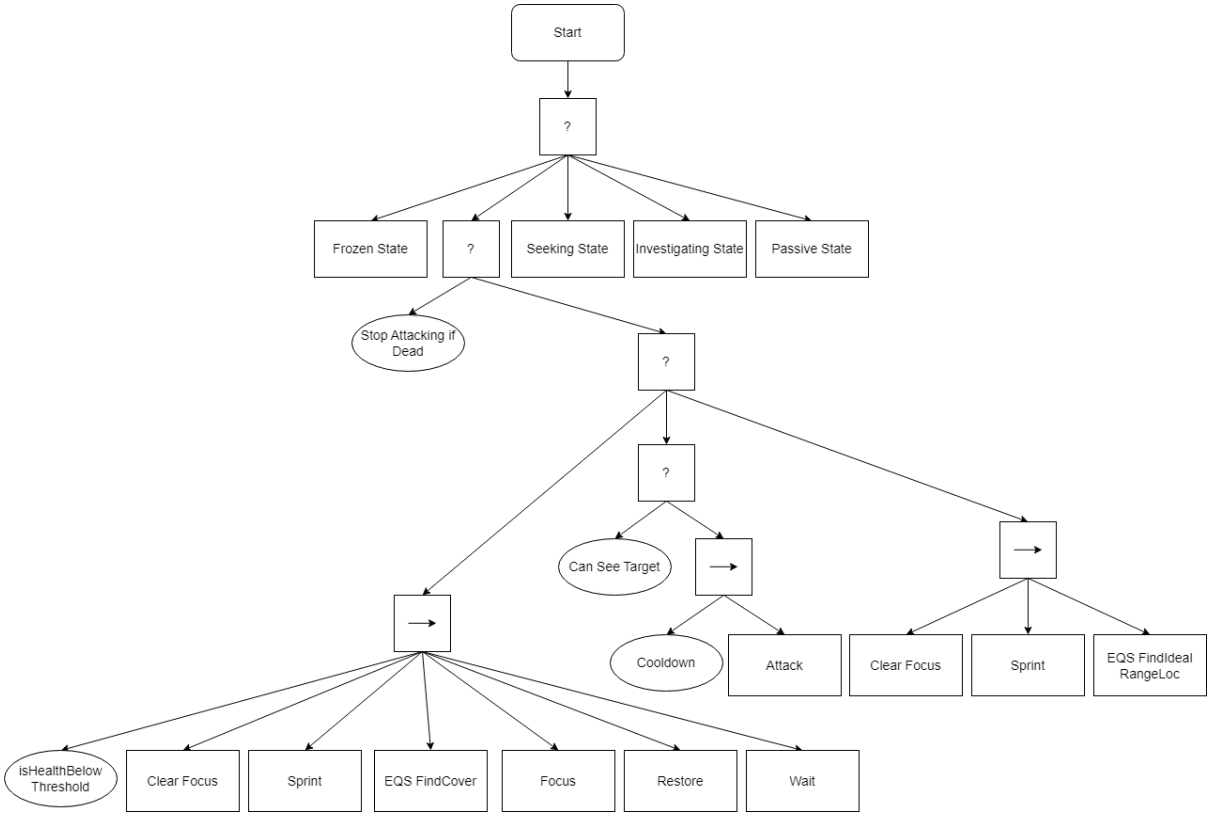


Gambar 3.6 *Behavior Tree Design* NPC *Skeleton* (*Sword*)

Behavior Tree ini menjalankan *Tasks* dalam jumlah yang lebih banyak sehingga menghasilkan kecerdasan buatan yang variatif. Pertama dimulai dengan pemeriksaan apakah NPC sedang menggunakan senjata atau tidak jika tidak, fokus ke *Attack Target* dan menjalankan *Task EquipWeapon* untuk mengeluarkan senjata dan melakukan pemeriksaan apakah *Attack Target* sedang menyerang atau tidak, jika menyerang maka NPC berhenti di tempat, fokus tetap kepada *Attack Target* dan jalankan *Task Block* untuk beberapa saat. Jika durasi *Cooldown* dari *Task Block* masih belum selesai maka, NPC akan menerima serangan dari *Attack Target* serta membatalkan semua *Task* lain dan hanya menjalankan *Task Frozen*. Kemudian NPC akan berusaha masuk ke dalam jarak sedikit jauh dari *Attack Target* untuk melakukan serangan melompat atau *JumpAttack* yang dilanjutkan langsung

dengan *SpinAttack* atau serangan berputar. Untuk serangan *JumpAttack*, NPC bisa melakukannya berulang kali jika *Attack Target* berada di jarak yang sudah ditentukan, sedangkan *SpinAttack* memiliki *Cooldown* selama 15 detik, sehingga serangan NPC tidak terlalu kuat dan juga tidak terlalu lemah, setelah itu NPC akan menyerang dengan serangan biasa dan mulai *Strafe* atau mengitari *Attack Target*. Jika NPC kehilangan pandangan dari *Attack Target* maka NPC akan menjalankan *subtree Investigating* dan jika tetap belum menemukan *Attack Target* dalam durasi *subtree Investigating*, maka NPC akan menyimpan senjatanya dan menjalankan *subtree Passive/Patrol State.*

3.3.2.2 NPC *Skeleton* (*Bow*)



Gambar 3.7 *Behavior Tree Design* NPC *Skeleton* (*Bow*)

Untuk *Behavior Tree Design* NPC *Skeleton* (*Bow*)*,* prioritas setelah *Frozen State* adalah menjalankan *Sequence FindCover, FindCover* akan berjalan jika *Decorator isHealthBelowThreshold* sudah terpenuhi yaitu jumlah Health adalah 20% kebawah. Jika terpenuhi maka NPC akan menghilangkan fokusnya dari *Attack Target* kemudian berlari ke tempat persembunyian dengan menjalankan *EQS FindCover*. Setelah sampai ke tempat persembunyian NPC akan fokus lagi ke *Attack Target* dan menjalankan *Task Restore* selama 1 detik. Karena fokus masih kepada *Attack Target*, setelah *Restore,* NPC akan keluar dari tempat persembunyian dan menjalankan *Decorator CanSeeTarget* untuk memeriksa apakah *Attack Target* terlihat kemudian melakukan serangan serta *strafe* atau mengitari *Attack Target* dan berusaha untuk menjaga jarak dari *Attack Target* dengan menjalankan *EQS FindIdealRangeLoc.* Jika *Attack Target* berada di luar jarak serangan NPC, NPC akan

berlari berusaha memasuki jarak serangan sesuai *EQS FindIdealRangeLoc.* Jika NPC kehilangan pandangan dari *Attack Target* maka NPC akan menjalankan *subtree Investigating* dan jika tetap belum menemukan Attack Target dalam durasi *subtree Investigating*, maka NPC akan menjalankan *subtree Passive/Patrol State.*

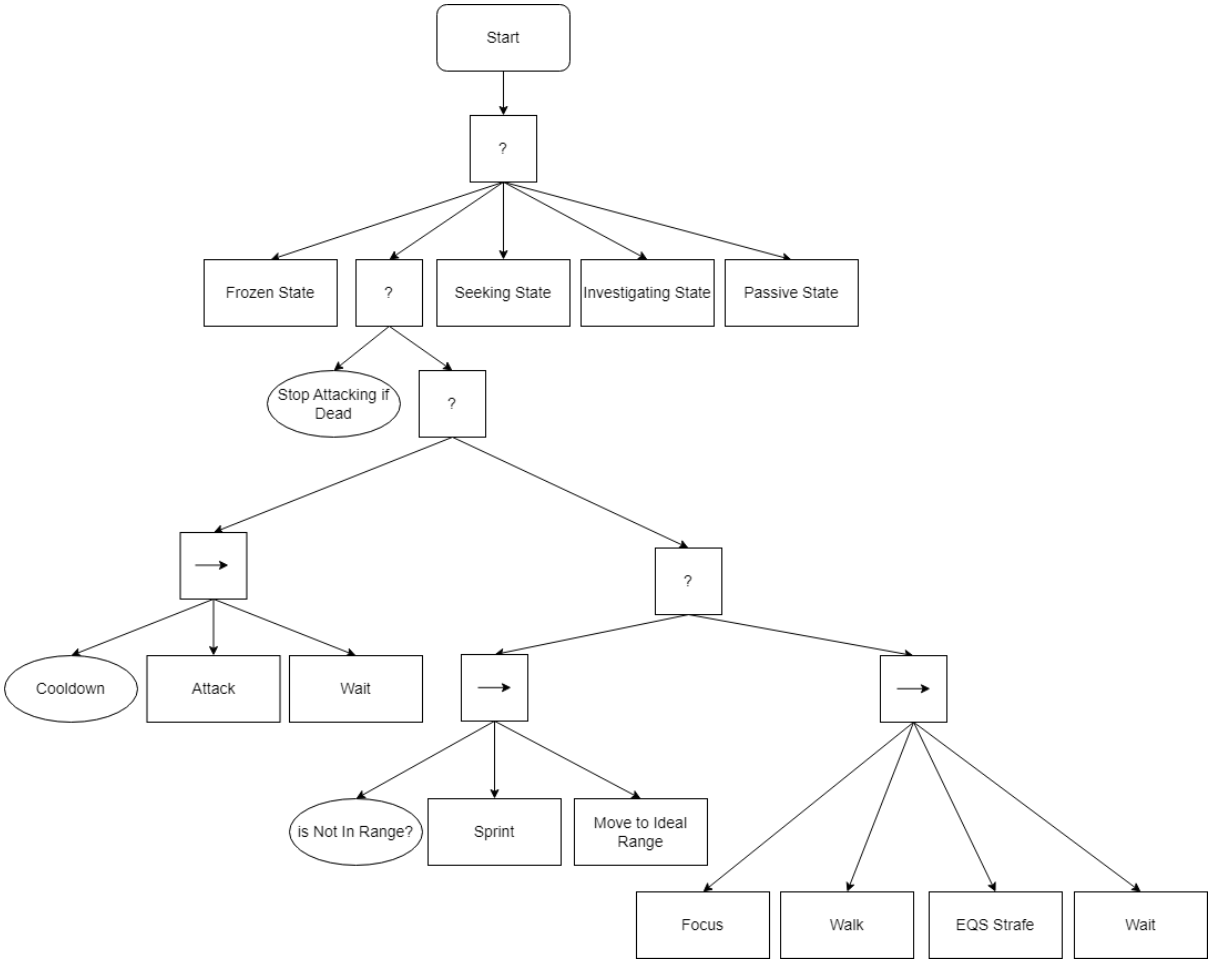
3.3.2.3 NPC *Ghoul*

Gambar 3.8 *Behavior Tree Design* NPC *Ghoul*

Ketika NPC melihat *Attack Target* atau *Player*, akan menjalankan *sequence Escape* Dimana NPC akan lari mencari tempat persembunyian dengan menjalankan *EQS FindCover* dan menunggu di tempat tersebut untuk beberapa saat, tetapi jika NPC mengetahui bahwa *Attack Target* sedang menyerang, maka akan menjalankan *sequence Attack* atau menyerang dalam 2 *Node* yang dibagi menjadi *Attack* dan *Strafe*. Pada bagian *Attack,* NPC menjalankan *Task Attack* setiap *Cooldown* 2 detik, dan selama *Cooldown* menjalankan *Strafe* dan dilakukan pemeriksaan apakah NPC berada di dalam jarak ideal dari *Attack Target*, jika tidak lakukan *Sprint* ke jarak ideal tersebut, jalankan *EQS Strafe* dan mulai mengitari *Attack Target* setiap durasi dari *Task Wait. Combat State* juga diberikan sebuah

Service bernama StopAttackingifDead, *Service* digunakan untuk melakukan pemeriksaan dan memperbarui *Blackboard. StopAttackingifDead* digunakan untuk melakukan pemeriksaan apakah *Attack Target* sudah masuk dalam keadaan *Dead* jika sudah, maka NPC akan berhenti menyerang dan masuk ke *Passive State.* Jika NPC kehilangan pandangan dari *Attack Target* maka NPC akan menjalankan *subtree Investigating* dan jika tetap belum menemukan Attack Target dalam durasi *subtree Investigating*, maka NPC akan menjalankan *subtree Passive/Patrol State.*

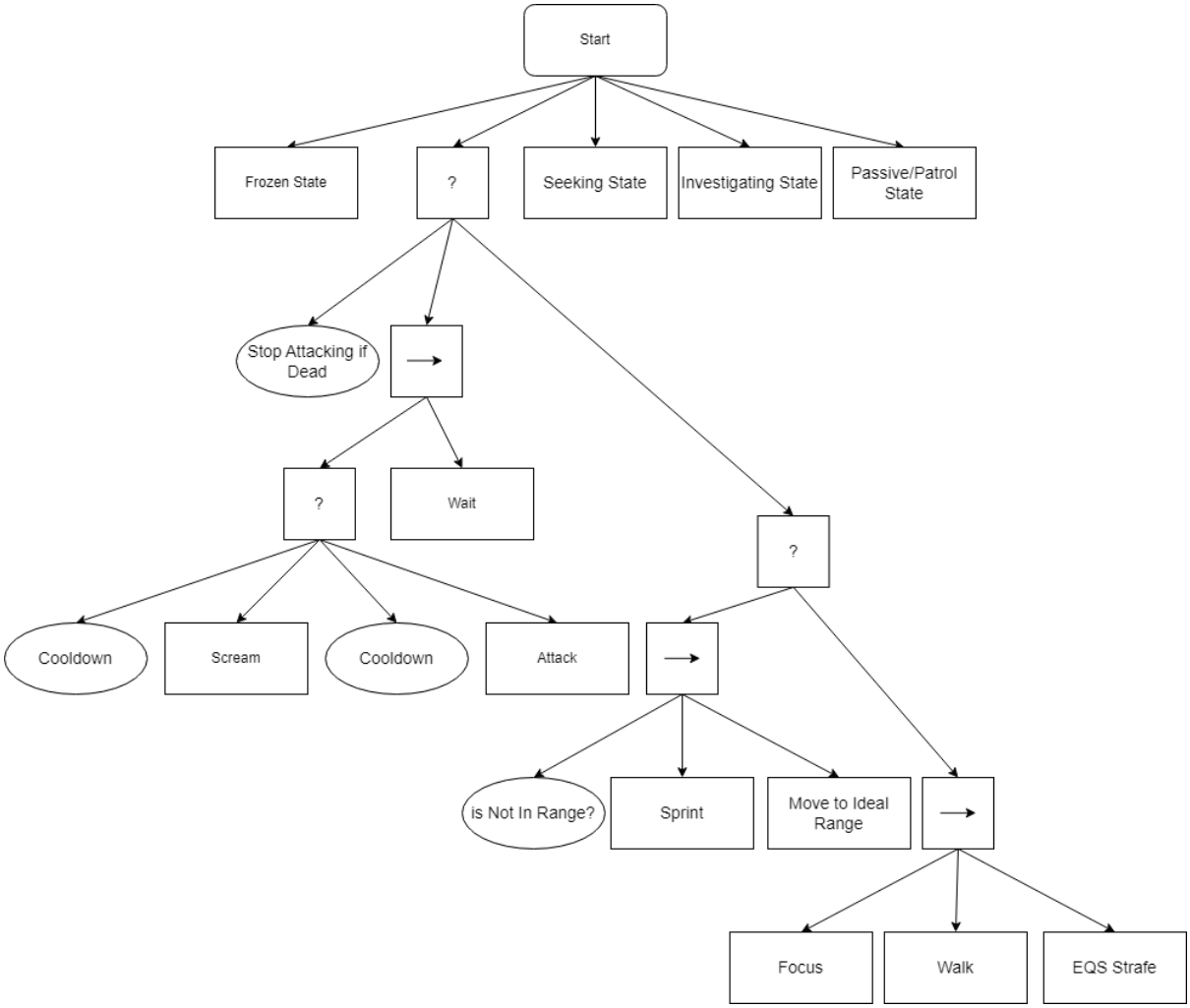
3.3.2.4 NPC *Goblin*



Gambar 3.9 *Behavior Tree Design* NPC *Goblin*

Untuk NPC Goblin ketika melihat Attack Target atau Player akan langsung berusaha mengejar dan menyerang pertama menjalankan sequence menyerang dengan menjalankan *Task Attack* setelah selesai menjalankan *Task Wait* dalam durasi singkat, ketika *sequence Attack* masih dalam *cooldown*, maka akan menjalankan *sequence* yang mengecek apakah *Attack Target* masih dalam jarak serang, jika tidak, maka NPC akan berusaha untuk lari masuk ke dalam jarak serang dan menjalankan *sequence* untuk melakukan *strafe* atau mengelilingi *Attack Target* dengan fokus pandangan ke *Attack Target* dan berjalan dengan menjalankan *EQS Strafe.*

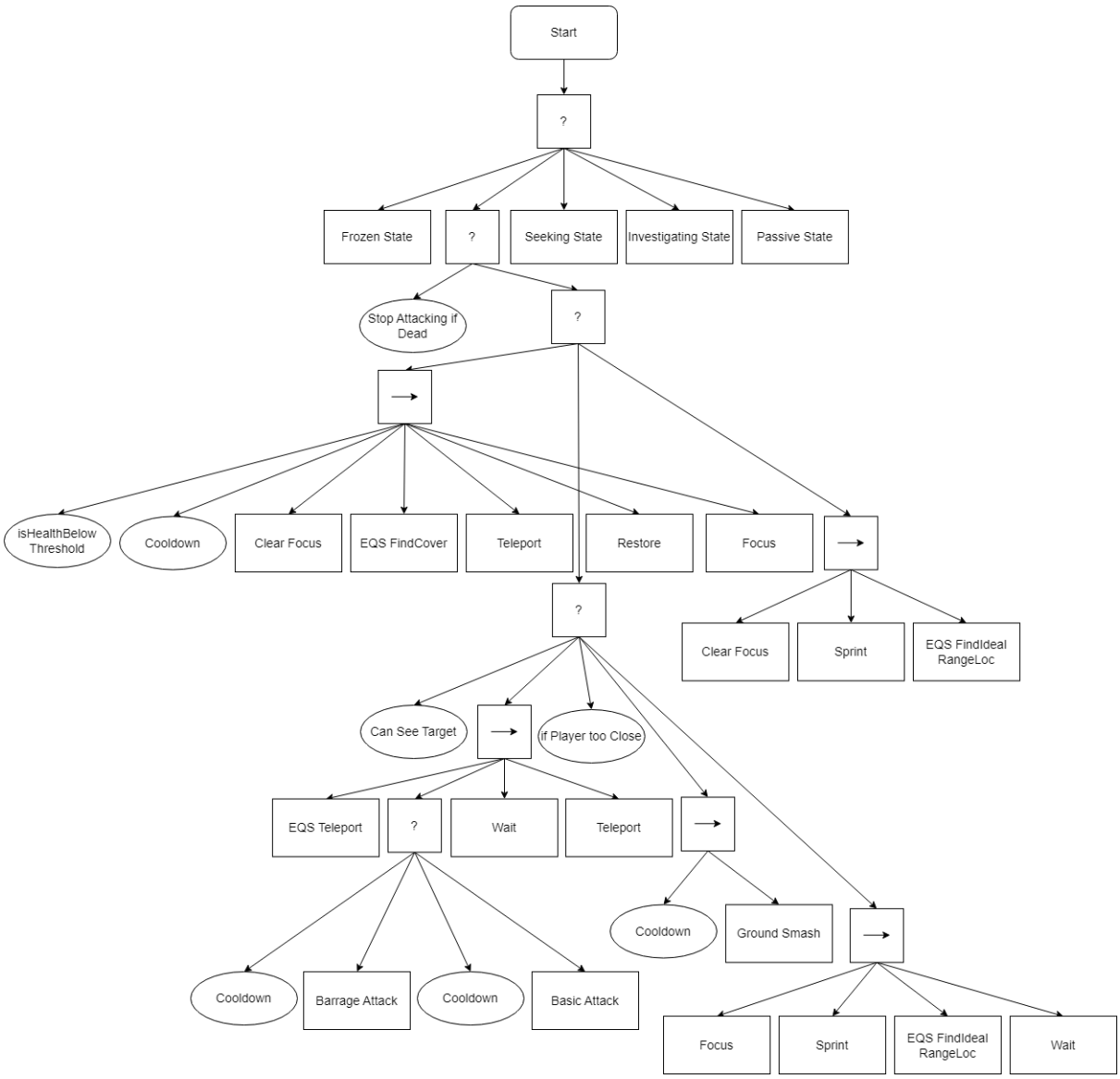
3.3.2.5 NPC *Zombie*



Gambar 3.10 *Behavior Tree Design* NPC *Zombie*

NPC Zombie memiliki kemampuan dimana memiliki kemampuan untuk melakukan serangan yang tidak memiliki *damage* bernama *Scream*. *Scream* adalah sebuah serangan dimana *Zombie* akan berteriak dengan suara yang besar memanggil NPC yang berada di sekitar radius teriakan tersebut dan dari teriakan tersebut NPC lain akan langsung berlari ke sumber teriakan tersebut dan jika melihat *Attack Target* atau *Player* maka akan langsung membantu *Zombie* untuk menyerang bersama. Setelah *Scream* berakhir akan dilanjutkan dengan serangan *Attack* dan dilanjutkan dengan *selector* apakah NPC masih berada di jarak *Attack*, jika masih maka akan menjalankan *sequence Strafe* untuk mengelilingi *Attack Target*. *Scream* memiliki *cooldown* yang cukup lama sehingga tingkat kesulitan dari permainan tidak menyusahkan Pemain.

3.3.2.6 NPC *Lich*



Gambar 3.11 *Behavior Tree Design* NPC *Lich*

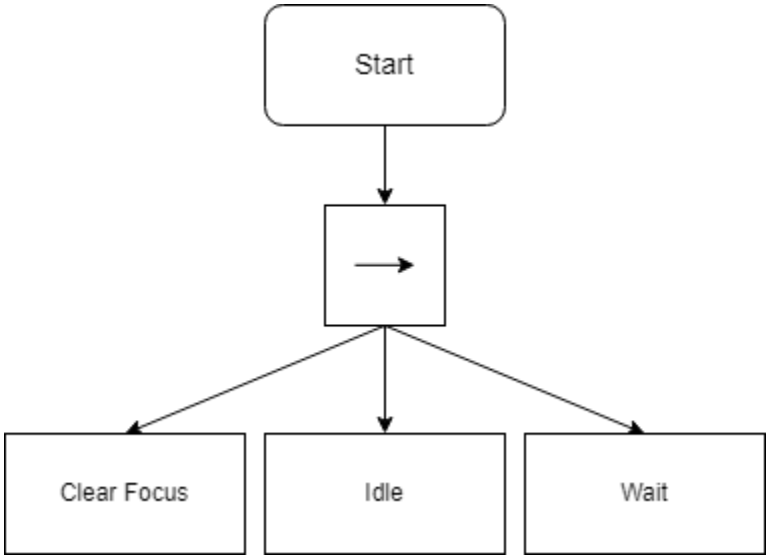
Sama seperti NPC *Skeleton* (*Bow*), NPC *Lich* juga melakukan pemeriksaan jumlah *Health* sebagai prioritas kedua setelah *Frozen State*, yang membedakan adalah *Lich* akan melakukan teleportasi ke titik di sekitar *Attack Target* setelah itu *Lich* akan melakukan serangan *Barrage Attack,* setelah itu dalam durasi yang singkat, *Lich* akan teleportasi lagi dan menjalankan *Basic Attack.* Ketika *Attack Target* berada di radius dekat *Lich*, maka *Lich* akan menyerang dengan menggunakan *GroundSmash.* Proses *Attack* dan *Groundsmash* hanya akan batal jika sedang melakukan *Heal*. Ketika *Lich* berada diluar jarak menyerang, maka harus masuk ke jarak ideal dengan menjalankan *EQS FindIdealRangeLoc* agar bisa menyerang kembali atau jika *Lich* kehilangan pandangan dari *Attack Target* maka akan menjalankan *Investigating State* atau *Attack Target* sudah dalam keadaan *Dead.*

Ketika *Health* dari *Lich* berjumlah 20%, *Lich* akan mencari tempat persembunyian dan menjalankan *Task Restore* untuk mengisi Healthnya kembali, setelah itu kembali fokus menyerang *Attack Target.*

3.3.3 *Behavior Sub-Tree Design*

Sebuah *Behavior Tree* bisa digunakan untuk menjalankan *Behavior Tree* lain di dalamnya. *Behavior Tree* lain ini hanya menjalankan *Tasks* yang sederhana serta membantu Penulis dalam membaca dan mengatur *Behavior Tree* utama.

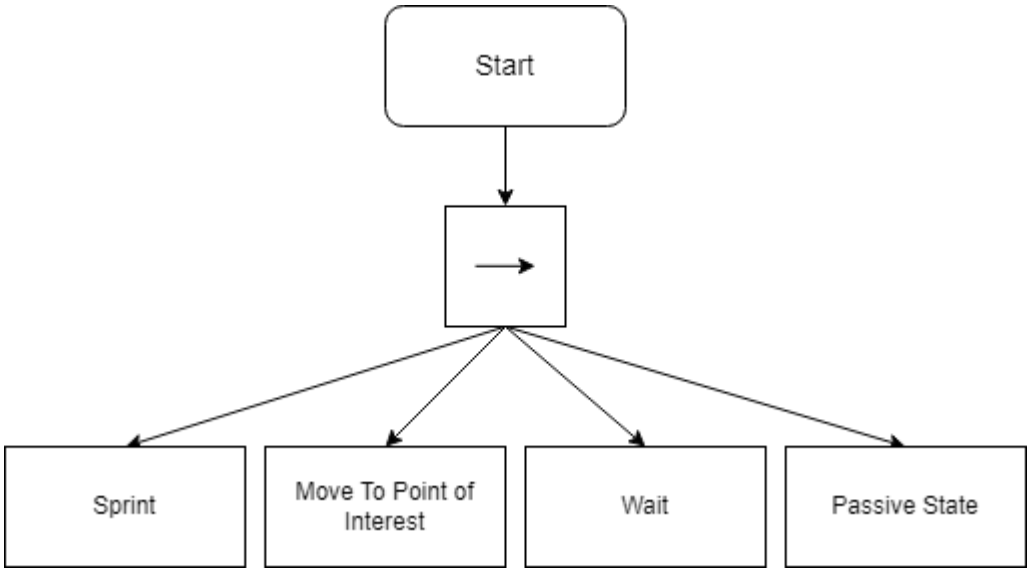
3.3.3.1 *Frozen State Sub-Tree*



Gambar 3.12 *Frozen Behavior Sub-Tree Design*

Frozen berjalan dengan *ClearFocus* untuk menghilangkan fokus terhadap *Target* serta mengubah mode pergerakan menjadi *Idle* atau diam ditempat selama 5 detik.

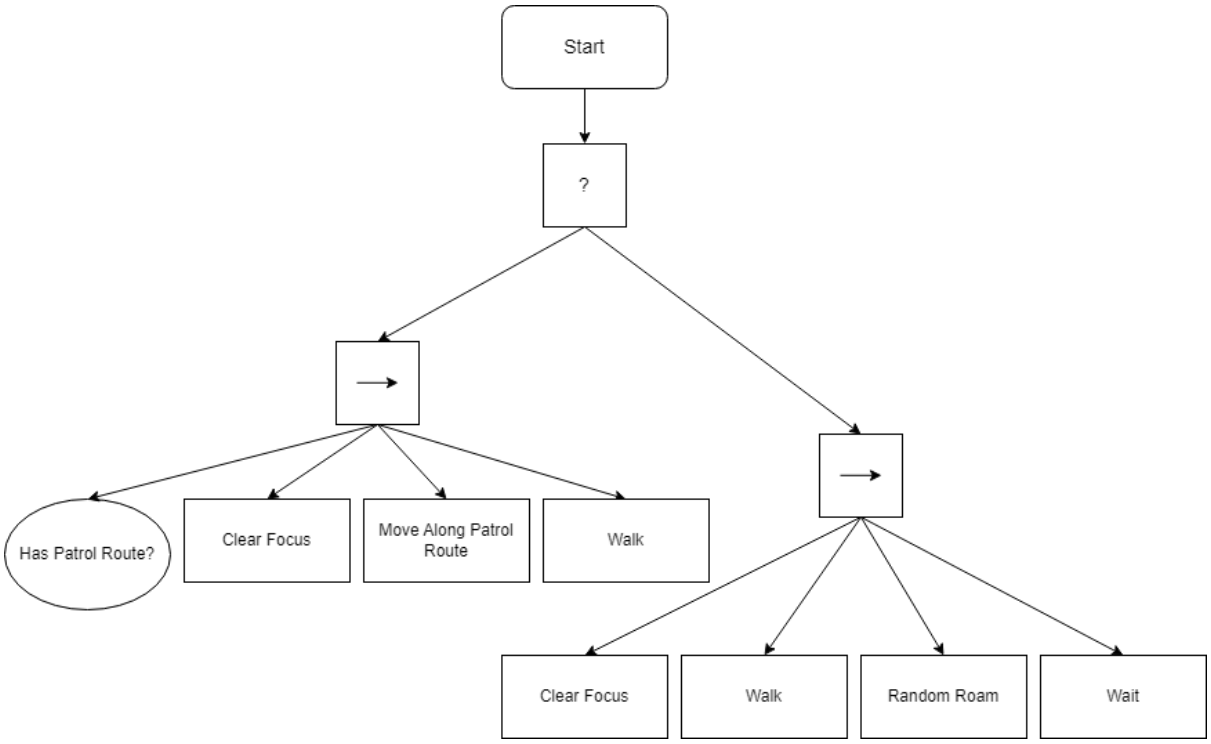
3.3.3.2 *Investigate State Sub-Tree*



Gambar 3.13 *Investigate Behavior Sub-Tree Design*

Investigate berjalan dengan mengatur mode pergerakan menjadi *Sprinting* ke arah *PointOfInterest* kemudian tetap berada di *PointOfInterest* selama 3 detik, setelah itu masuk ke *Passive State*.

3.3.3.2 Passive State Sub-Tree



Gambar 3.14 *Passive Behavior Sub-Tree Design*

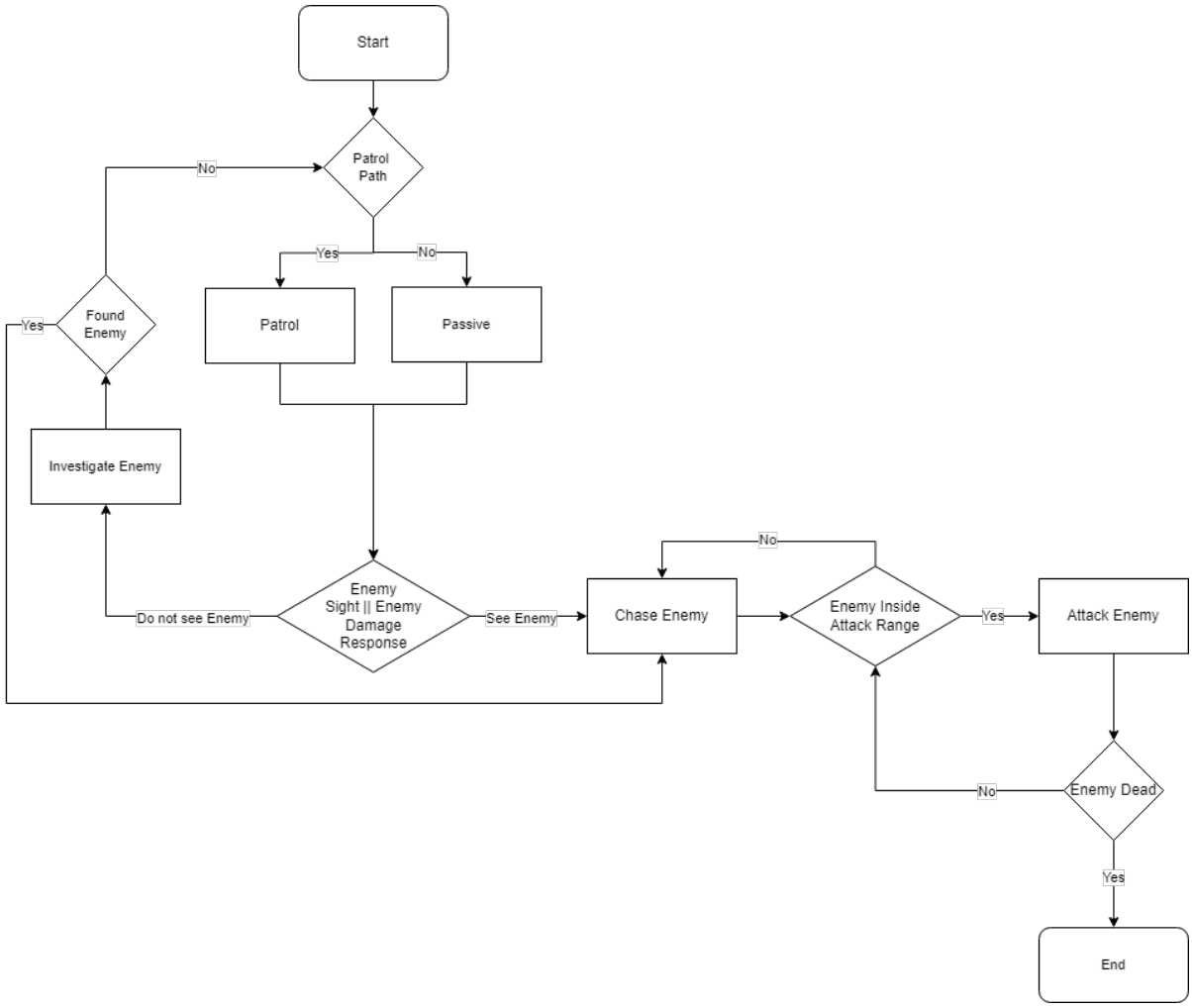
Passive dibagi menjadi 2, yang pertama dilakukan pengecekan dengan *Decorator* apakah memiliki jalur patrol atau *Has Patrol Route*, jika ada maka menjalankan *Task MoveAlongPatrolRoute* dan atur mode pergerakan menjadi *Walk.* Jika tidak ada jalur patroli, maka NPC akan berjalan ke tujuan yang acak dalam radius tertentu.

3.3.3.2 *Seeking State Sub-Tree*

Gambar 3.15 *Seeking Behavior Sub-Tree Design*

Seeking menjalankan *Tasks* dengan jumlah yang cukup banyak, pertama menghapus fokus NPC dari *Target*, mengubah mode pergerakan menjadi *Sprint* ke arah *PointOfInterest* dan menunggu selama 1 detik. Kemudian masuk ke *sequence Seek* dengan menjalankan EQS_Seeking dan berjalan di daerah sekitar *PointOfInterest,* fokus diubah ke diri NPC sendiri agar mengenal posisinya sekarang, setelah 1 detik sesuai dengan *Decorator Loop* 5 kali, berarti *Sequence Seek* akan dijalankan sebanyak 5 kali. Setelah itu *ClearFocus* dan menjalankan *task SetStateAsPassive.*

3.3.4 Implementasi Behavior Tree Terhadap AI



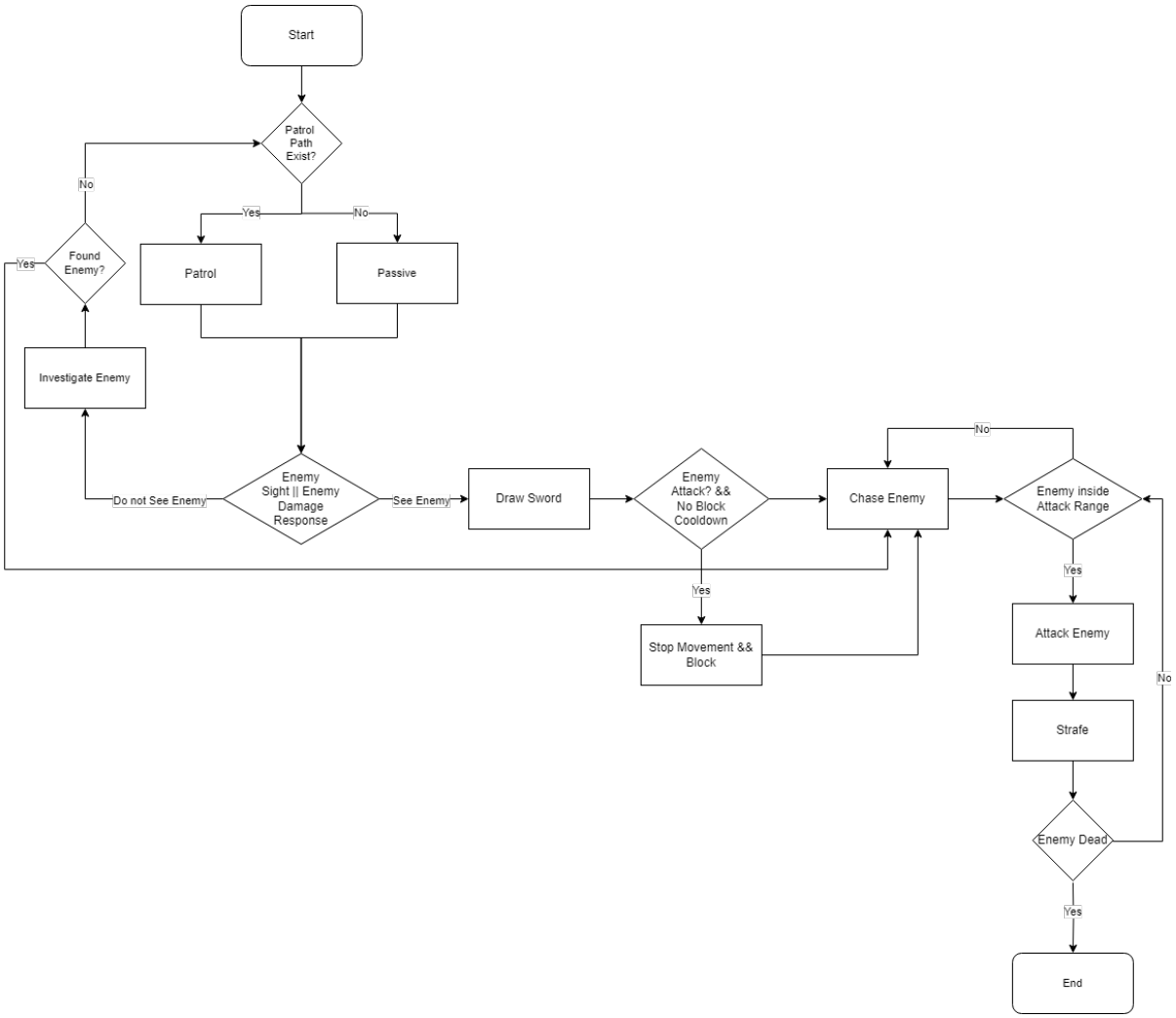
Gambar 3.16 *Flowchart* implementasi *Behavior Tree* terhadap NPC (*enemy*)

Flowchart pada Gambar 3.5 merupakan fondasi atau dasar dari *Behavior Tree* yang digunakan oleh semua jenis NPC yang ada di *game* ini dengan beberapa perubahan atau tambahan. Awalnya NPC melakukan pengecekan apakah memiliki *Patrol Path* atau jalur patroli. Jika ada, NPC akan melakukan *patrol* sesuai dengan titik yang sudah ditentukan, jika tidak ada NPC akan bersifat *passive* atau berjalan ke titik tujuan secara acak. Jika NPC berhasil melihat *Player* atau dari sudut pandang NPC dianggap sebagai *enemy* dari radius pandangannya atau berhasil merespon arah dari *damage* yang diterima, NPC akan mulai mengejar *Player*. Jika *Player* sudah berada di dalam radius serang milik NPC, NPC akan langsung menjalankan fungsi *Attack* kepada *Player.* Jika *Health* dari *Player* habis atau berjumlah 0, *enemy* akan berhenti menyerang dan permainan akan diarahkan ke halaman *game over.*

Jika *Player* berhasil melarikan diri dari serangan enemy atau keluar dari radius serangannya, *enemy* akan berusaha mengejar *Player* kembali, jika *Player* berhasil keluar dari radius penglihatan *enemy, enemy* akan menjalankan fungsi *Investigate* didalam radius terakhir kali *enemy* melihat *Player*, jika berhasil, *enemy* kembali mengejar *Player*, jika tidak, *enemy* akan melakukan pengecekan apakah

memiliki *patrol path* atau tidak. Jika *enemy* menerima *damage* dari *Player* diluar radius penglihatannya*, enemy* akan menjalankan fungsi *investigate.*

3.3.4.1 Implementasi *Behavior Tree* Terhadap *AI Skeleton (Sword)*

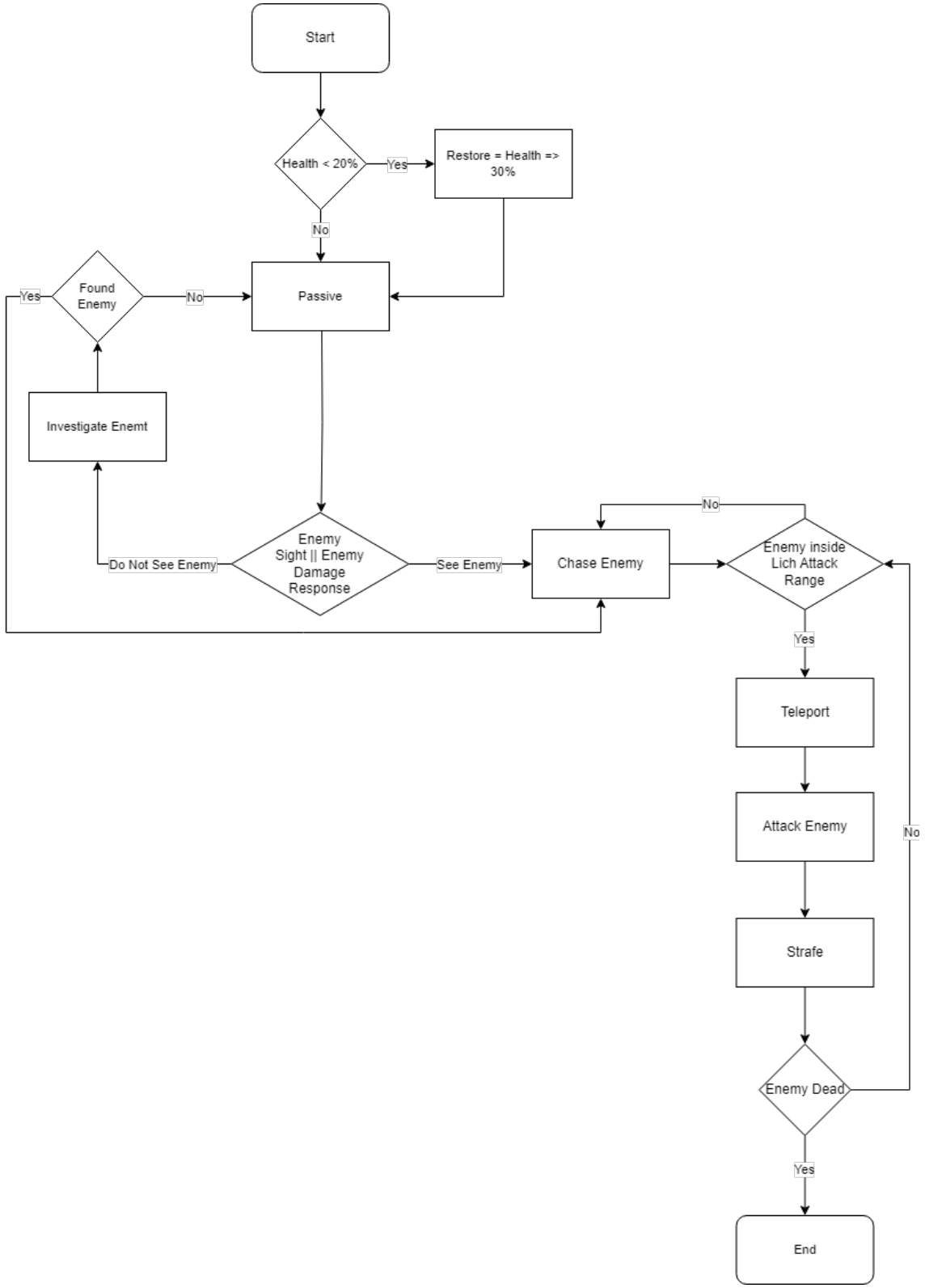


Gambar 3.17 *Flowchart* Implementasi *Behavior Tree* Terhadap NPC (*enemy*) *Skeleton (Sword)*

Pada dasarnya implementasi Behavior Tree untuk NPC *Skeleton* memiliki penjelasan yang sama seperti di sub bab sebelumnya yaitu 3.3.2, tetapi yang membedakan adalah ketika *Skeleton* melihat *Player* atau menerima *damage* dari *Player*, *Skeleton* tidak langsung mengejar tetapi akan mengeluarkan senjata terlebih dahulu Kemudian bergerak menuju posisi *Player.* Tetapi ketika *Skeleton* menerima serangan dari *Player*, *Skeleton* akan memeriksa apakah kemampuan *Block*-nya sedang dalam cooldown atau tidak, jika tidak Skeleton akan berhenti di posisinya dan menahan serangan dari *Player* untuk beberapa saat dan lanjut bergerak ke posisi *Player,* tetapi jika *Block* sedang *cooldown*, *Skeleton* akan menerima serangan dan lanjut bergerak ke posisi *Player*.Setelah sampai di dekat *Player, Skeleton* akan menyerang satu kali setelah itu akan masuk ke fungsi *Strafe* untuk mulai mengitari *Player* untuk beberapa saat dan menyerang *Player* kembali. Jika Skeleton kehilangan pandangan dari

Player, Skeleton akan menginvestigasi untuk beberapa saat dan jika tidak menemukan *Player,* maka *Skeleton* akan menyimpan senjatanya dan memeriksa apakah memiliki jalur *patrol* atau tidak.

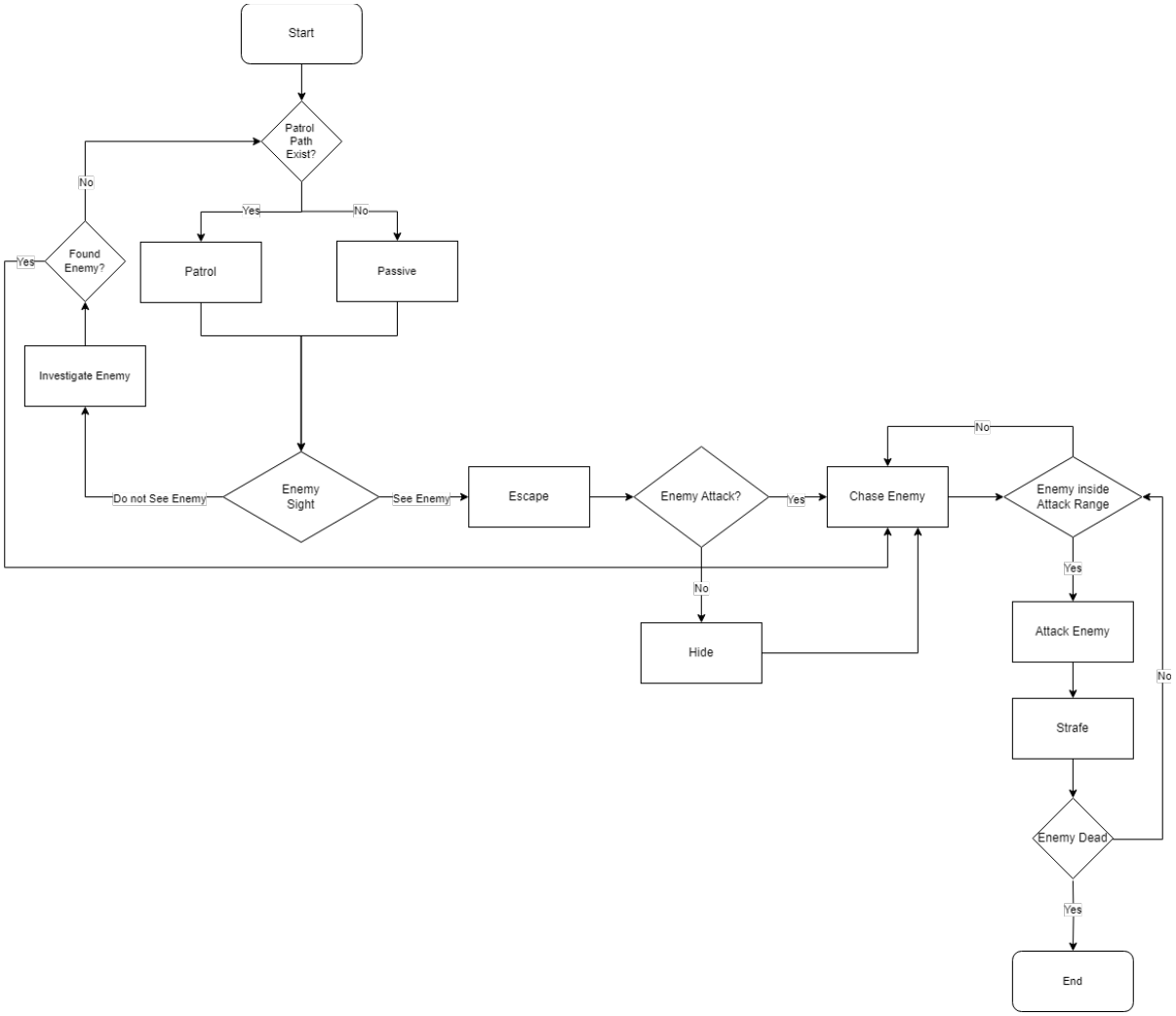
3.3.4.2 Implementasi *Behavior Tree* Terhadap *AI Lich*



Gambar 3.18 *Flowchart* Implementasi *Behavior Tree* Terhadap NPC (*enemy*) *Lich*

Lich akan dimulai dengan memeriksa apakah jumlah *Health* sekarang dibawah 20%, jika dibawah 20% *Lich* akan menggunakan kemampuan *Restore* untuk mengisi kembali *Health* hingga mencapai 30% dari *Maximum Health* yang dimiliki. Kemudian jika Lich melihat atau menerima *damage* dari *Player, Lich* akan mengejar *Player* sampai berada di dalam radius serangannya dan Lich akan langsung menggunakan kemampuan *Teleport* untuk berpindah posisi dengan cepat ke titik buta dari *Player* yaitu di belakangnya. *Lich* kemudian melakukan serangan, masuk ke fungsi *Strafe* dan kembali menggunakan kemampuan *Teleportnya*. Karena *Lich* tidak memiliki jalur *patrol,* ketika kehilangan *Player* setelah menginvestigasi untuk beberapa saat, *Lich* akan masuk ke mode *passive.*

3.3.4.3 Implementasi *Behavior Tree* Terhadap *AI Ghoul*

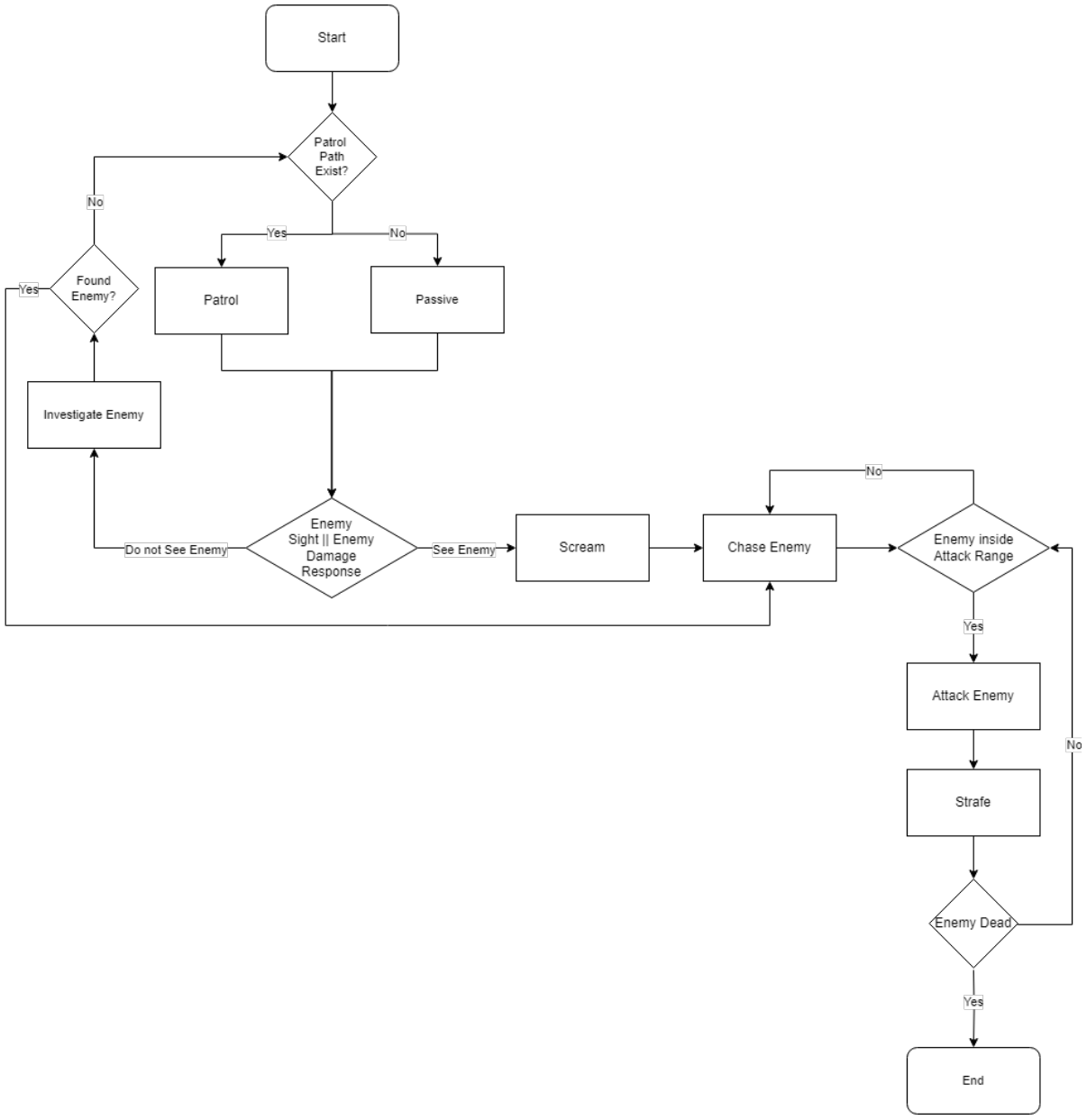


Gambar 3.19 *Flowchart* Implementasi *Behavior Tree* Terhadap NPC (*enemy*) *Ghoul*

Behavior dari *Ghoul* akan dimulai dari pemeriksaan apakah memiliki jalur patroli atau tidak, jika tidak maka akan menjalankan *Task Passive* dimana NPC akan berjalan secara acak di dalam *stage.* Ketika *Ghoul* melihat musuh atau *Attack Target* yang pertama dilakukan adalah melarikan diri dan

mencari tempat persembunyian, tetapi ketika dalam pelarian *Ghoul* mendeteksi bahwa *Attack Target* melakukan serangan maka *Ghoul* tidak jadi melarikan diri dan mulai mengejar *Attack Target* dan berusaha menyerang.

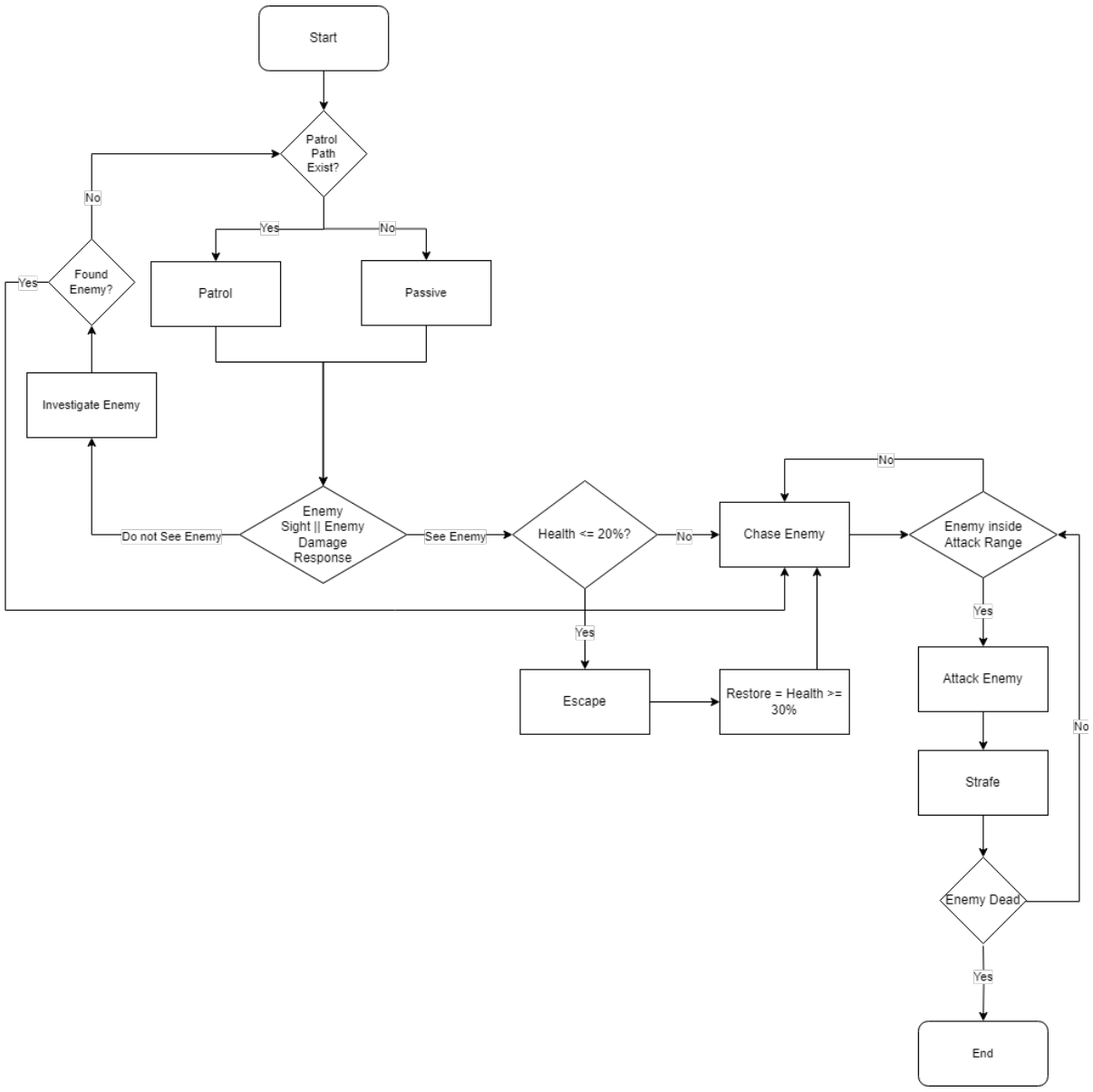
3.3.4.4 Implementasi *Behavior Tree* Terhadap *AI Zombie*



Gambar 3.20 *Flowchart* Implementasi *Behavior Tree* Terhadap NPC (*enemy*) *Zombie*

Untuk *Behavior* dari *Zombie* pertama akan melakukan pemeriksaan apakah memiliki jalur patroli atau tidak, jika ada maka menjalankan *Task Patrol* jika tidak menjalankan *Task Passive.* Ketika *Zombie* melihat atau menerima serangan dari musuh atau *Attack Target* maka hal pertama yang dilakukan adalah menjalankan *Task Scream* yang akan memanggil NPC dalam radius *Scream.* Kemudian *Zombie* akan berusaha mengejar dan menyerang *Attack Target*.

3.3.4.5 Implementasi *Behavior Tree* Terhadap *AI Skeleton (Bow)*



Gambar 3.21 *Flowchart* Implementasi *Behavior Tree* Terhadap NPC (*enemy*) *Skeleton (Bow)*

Behavior dari Skeleton (Bow) yang membedakan dari yang lain adalah ketika NPC melihat musuh atau Attack Target pertama akan memeriksa apakah jumlah Health yang dimilikinya sekarang kurang atau sama dengan 20% jika tidak maka akan mengejar dan menyerang Attack Target. Tetapi jika Health kurang atau sama dengan 20% maka NPC akan membatalkan Task yang mungkin sedang dilakukan seperti mengejar atau menyerang Attack Target dan berlari dari Attack Target untuk berusaha mencari tempat persembunyian dan menjalankan Task Restore untuk mengisi Health sampai dengan lebih atau sama dengan 30%. Setelah itu kembali mengejar dan menyerang Attack Target.

3.4 *Interface* Desain

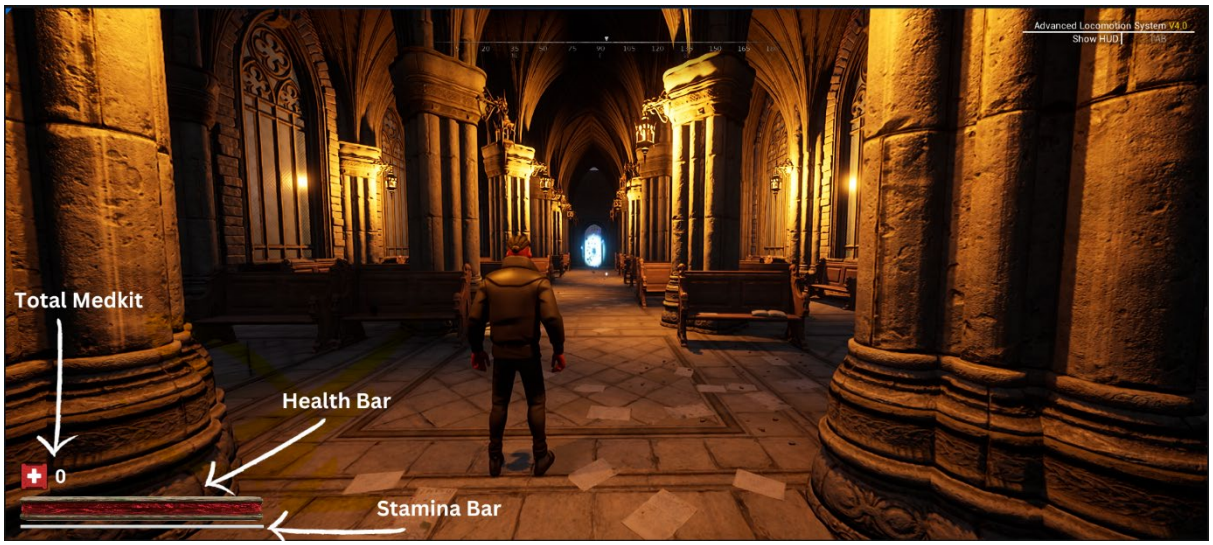
Pada sub bab ini, akan dijelaskan *interface* yang dimiliki oleh *game* ini.

3.4.1 Main Menu

Gambar 3.22 Tampilan *interface Main Menu*

Ketika permainan dimulai, *Player* akan dibawa ke *interface* awal seperti gambar 3.6. *Menu Play Game* akan membawa *Player* ke *Stage Tutorial, Menu Credits* akan membawa Player ke tampilan *Credits* dan *Exit* akan menutup permainan.

3.4.2 *Playing*



Gambar 3.23 Tampilan *interface* ketika di dalam *game*

Setelah *Player* memilih menu *Play Game,* permainan beralih dari *Main Menu* ke *Stage Tutorial* yang memiliki tampilan *interface* seperti gambar 3.7 sesuai keterangan sudah ada tampilan untuk jumlah *Medkit*, *Health Bar* dan *Stamina Bar.*

3.4.3 *Inventory*

Gambar 3.24 Tampilan *interface Inventory*

Setelah *Player* menemukan sebuah senjata, senjata akan langsung ditampilkan seperti gambar 3.8, tampilan angka dari *Mag Ammo*, *Bag Ammo* dan *total Ammo* akan otomatis diperbarui ketika *Player* menemukan *Ammo* tergantung dari jenis senjata yang sudah diperoleh atau ketika *Player* sedang menembak.

3.4.4 *Game Over*

Gambar 3.25 Tampilan *Game Over*

Ketika jumlah *Health* dari *Player* mencapai 0 atau habis, maka Player dianggap kalah dan masuk ke *interface Game Over* seperti di gambar 3.9. Menu *Restart* untuk mengulangi permainan dari *Stage Tutorial* dan *Main Menu* untuk kembali ke *interface* awal ketika game dijalankan.