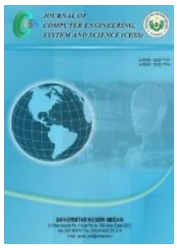
Cess
(Journal of Computing Engineering, System and Science) 8(2) July 2023 542-553

e-ISSN: 2502-714x
p-ISSN: 2502-7131

Contents list available at www.jurnal.unimed.ac.id

CESS
(Journal of Computing Engineering, System and Science)

journal homepage: https://jurnal.unimed.ac.id/2012/index.php/cess



Pengembangan *Finite State Machine Agent* Pada *Game* Idle Breeder

Development of Finite State Machine Agent in Idle Breeder Game

Agung Riyadi[1*], Sultan Ilyas Arsalillah Yuswan Syah[2]

[1,2] Politeknik Negeri Batam
Batam Centre, Jl. Ahmad Yani, Tlk. Tering, Kec. Batam Kota, Kota Batam, Kepulauan Riau 29461, Indonesia
email: [1]sultanilyasarsal@gmail.com, [2] agung@polibatam.ac.id

A B S T R A K

"*Idle Breeder*" merupakan sebuah *game idle top down* yang memberikan pengalaman berternak dengan suasana santai bagi para pemainnya. Setiap karakter hewan didesain dengan perilaku yang menyerupai hewan aslinya. Namun, penerapan *game* sebelumnya menghadapi berbagai masalah, terutama dalam manajemen kode program yang menyulitkan developer secara *scalable* dan mengakibatkan beberapa komponen tidak dapat digunakan kembali (*reusable*). Oleh karena itu, penelitian ini bertujuan untuk mengatasi masalah tersebut dengan mengimplementasikan *Finite State Machine* (FSM) sebagai solusi. Metode pengembangan yang digunakan adalah *Multimedia Development Life Cycle* (MDLC), sesuai dengan pengembangan fitur ini. Temuan dari penelitian ini akan memberikan contoh implementasi FSM pada *game agent*, membantu para pengembang game mobile memahami kapan dan bagaimana mengimplementasikan FSM secara tepat. Hasil pengujian menunjukkan beberapa kelebihan FSM seperti penggunaan *script* yang *modular* dan *reusable*. Pendekatan ini mengurangi redundansi dalam kode dan mempercepat proses implementasi. Selain itu, FSM juga menawarkan skalabilitas tinggi, memungkinkan pengembangan game dalam jangka waktu yang panjang dan mempermudah *maintenance*. Namun, dikarenakan menggunakan FSM, performa menjadi lebih berat. Oleh karena itu, penggunaan FSM akan lebih efektif ketika ingin memiliki skalabilitas tinggi dan sistem yang kompleks.

Kata Kunci: *Finite State Machine, Game Agent, Android, Mobile, Mobile Game.*

A B S T R A C T

"Idle Breeder" is a top down idle game that provides players with a relaxing farming experience. Each animal character is designed with behaviors that resemble the real animal. However, previous game implementations faced various problems, especially in program code management which made it difficult for developers to be scalable and resulted in some components not being reusable. Therefore, this research aims to overcome these problems by implementing Finite State Machine (FSM) as a solution. The development method used is

**Penulis Korespondensi:*
email: sultanilyasarsal@gmail.com

Multimedia Development Life Cycle (MDLC), in accordance with the development of this feature. The findings of this research will provide examples of FSM implementation in game agents, helping mobile game developers understand when and how to implement FSM appropriately. The test results show several advantages of FSM such as the use of modular and reusable scripts. This approach reduces redundancy in code and speeds up the implementation process. In addition, FSM also offers high scalability, allowing for long-term game development and easier maintenance. However, due to the use of FSM, the performance becomes slower. Therefore, using FSM is more effective when high scalability and complex systems are desired.
Keywords: *Finite State Machine, Game Agent, Android, Mobile, Mobile Game*

1. PENDAHULUAN

Perkembangan teknologi saat ini telah berkembang sangat pesat dan salah satunya adalah pengembangan *game*. *Game* merupakan salah satu media hiburan yang sering dipilih oleh banyak orang karena menyenangkan. Selain sebagai hiburan, *game* juga dapat digunakan sebagai media edukasi untuk memperluas wawasan pemainnya [1], [2]. Pembelajaran dengan menggunakan *game* sebagai media edukasi merupakan cara yang efektif dibandingkan dengan media lain seperti *website* atau *ebook*.

Banyak yang memiliki *smartphone* sebagai alat yang dapat membantu mereka dalam kehidupan sehari-hari, mulai dari anak-anak hingga orang dewasa. Anak-anak zaman sekarang bisa belajar dengan menggunakan *smartphone* atau perangkat *mobile* lainnya. Biasanya mereka belajar dengan menonton video atau bermain *game*. Hal ini menunjukkan banyak pengguna yang bermain *game* di *platform mobile*, terutama yang masih berusia muda.

"*Idle Breeder*" adalah sebuah *game idle top down* yang menawarkan pengalaman berternak dengan suasana yang santai bagi para pemainnya. Di dalam *game* ini, setiap karakter hewan didesain untuk memiliki perilaku yang menyerupai hewan aslinya. Dalam penerapan *game* sebelumnya, terdapat banyak masalah terutama dalam manajemen kode program yang membuat *game* sulit untuk dikembangkan lebih besar (*scalable*) dan beberapa komponen didalamnya tidak *reusable*. Oleh karena itu, penelitian ini akan membahas tentang pengimplementasian *Finite State Machine* (FSM) sebagai solusi untuk mengatasi masalah tersebut. FSM digunakan sebagai metode perancangan sistem kontrol yang mengatur *state* (keadaan), *event* (kejadian), dan *action* (aksi) karakter hewan dalam game [2], [3], [9], [10], [14], [18], [19]. Keputusan untuk menggunakan FSM didasari oleh kemampuannya dalam menangani sistem state yang kompleks dengan pendekatan yang modular, menyediakan skalabilitas, serta memungkinkan proses *debugging* dan *maintenance* yang lebih mudah. Metode pengembangan yang digunakan adalah *Multimedia Development Life Cycle* (MDLC), yang sesuai dengan pengembangan fitur ini. Temuan dari penelitian ini bertujuan untuk memberikan contoh implementasi FSM pada *game agent*, sehingga para pengembang *game mobile* dapat memahami cara dan kapan mengimplementasikan FSM secara tepat.

2. DASAR/TINJAUAN TEORI

2.1. Game Agent

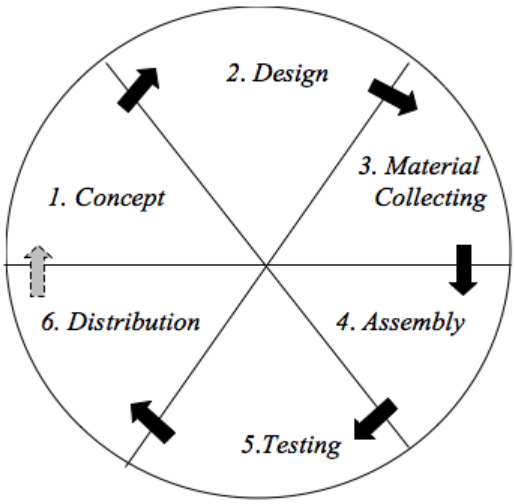
Game agent (Intelligent agent) merupakan *agent* yang secara cerdas mengontrol karakter *game* dengan menggunakan model logika *Finite State Machine* (FSM) [3]. Setiap aksi atau aktivitas dilakukan oleh *agent* untuk memenuhi kondisi lingkungannya [4].

2.2. Finite State Machine

Suatu karakter atau *agent* dibuat dengan cukup cerdas dengan mengimplementasikan metode *Finite State Machine* (FSM), yang merupakan model berdasarkan struktur data untuk menggambarkan *action* dengan urutan berbagai *event* [5]. FSM adalah *state machine* sederhana di mana keadaan yang berbeda dihubungkan oleh beberapa kondisi. Jika kondisi tertentu terpenuhi maka akan bertransisi dari satu *state* ke *state* lainnya [6], [7], [19], [21]. FSM adalah salah satu metode paling terkenal untuk memodelkan perilaku *agent* atau NPC dalam sebuah *game* [8]. Dalam metode FSM untuk merancang sistem kontrol yang menggambarkan prinsip kerja sistem menggunakan tiga hal yaitu *state, event, action* yang digunakan dalam *game agent* untuk dapat memberikan tindakan dan reaksi kepada pemain ketika *game* dimainkan [9], [13], [16], [17].

3. METODE

Metode pengembangan yang digunakan adalah metode *Multimedia Development Life Cycle* (MDLC) dari Luther-Sutopo [12], [16], [20]. MDLC yang digunakan untuk membuat aplikasi melalui enam tahapan antara lain: *concept*, *design, material collecting, assembly, testing*, dan *distribution* [11], [16], [19], [20].



Gambar 1. *Multimedia Development Life Cycle* [11], [12]

3.1. Concept

Tahap *concept* (konsep) adalah tahapan untuk menentukan tujuan, sasaran atau target pengguna program yang akan dibuat oleh peneliti. Selain itu juga menentukan macam aplikasi, tujuan aplikasi, dan lain-lain [11], [12], [16]. Pada tahapan ini, dilakukan pengonsepan apa yang akan diimplementasi pada *game agent*. Seperti kompleksitas *agent* tidak perlu tingkat lanjut dikarenakan *game casual* dan target audiens mulai dari anak-anak, maka dari itu sistemnya cukup sederhana sehingga dapat mudah dimengerti.

3.2. Design

Tahap *design* (perancangan) adalah tahap menyusun secara rinci spesifikasi mengenai arsitektur program dan hal yang berkaitan dengan pembuatan aplikasi [11], [12], [16]. Pada tahapan ini, membuat *class* diagram *game agent* dan desain model FSM untuk *game agent*.

3.3. Material Collecting

Tahap *material collecting* adalah tahap pengumpulan materi atau bahan yang sesuai dengan kebutuhan penelitian dan dapat dikerjakan secara paralel dengan tahap *assembly* [11], [12], [16]. Bahan-bahan yang diperlukan untuk pengembangan *game agent* adalah aset untuk *game* seperti 3D model karakter *game agent*, animasi karakternya, *sound effects* pada saat interaksi, 2D *sprite*, dan lain-lain.

3.4. Assembly

Tahap *assembly* (pembuatan) adalah tahap yang berdasarkan pada tahap desain yang dimana semua bahan yang didapat/dibuat dan perancangan yang telah disusun sebelumnya pada tahap desain akan diterapkan pada tahap ini yang sesuai untuk pembuatan aplikasi [11], [12], [16]. Pada tahap ini, akan diterapkan metode FSM pada *game agent* berdasarkan model dari tahap desain sehingga *agent* dapat bergerak dan berinteraksi sesuai dengan pengonsepan.

3.5. Testing

Tahap *testing* dilakukan saat tahap *assembly* atau pembuatan aplikasi telah selesai dengan menjalankan aplikasi atau program dan uji coba dan mengamati apakah terjadi adanya kesalahan pada program [11], [12], [16]. Pada tahap ini, ada tahap *alpha testing* yang dimana pengujian dilakukan oleh tim *developer* dan sekitarnya [12]. Selama *alpha testing* akan dilakukan *Black-box testing* yaitu pengujian perangkat lunak dari segi spesifikasi fungsional dengan tujuan untuk mengetahui apakah fungsi, masukan dan keluaran dari perangkat lunak sesuai dengan spesifikasi yang dibutuhkan tanpa menguji desain dan kode program [15], [20], [21]. Adapun *beta testing* yaitu *game* akan di uji coba oleh pengguna di luar selain tim selayaknya pemain pada dasarnya [12]. Selama *beta testing* akan dilakukan *playtest* ke beberapa *user*, lalu *user* akan melaporkan jika terjadi kesalahan (*bug*). Selanjutnya, kesalahan tersebut akan didaftarkan kemudian diperbaiki satu per satu selama pengembangan sampai semua kesalahan tidak muncul lagi.

3.6. Distribution

Tahapan terakhir yaitu aplikasi atau program yang telah dibuat akan disimpan dalam suatu media penyimpanan [11], [12], [16]. Distribusi aplikasi menyesuaikan *platform* yang dipilih. Pada penelitian ini, setelah sistem FSM pada *game agent* sudah selesai dibuat, maka selanjutnya akan diimplementasikan langsung ke dalam inti *game*. *Game* yang dibuat merupakan *mobile game* untuk *platform Android*, maka dari itu di-*export* dengan cara di-*compile* dan di-*build* sehingga menghasilkan *file* dengan format *.apk.

4. HASIL DAN PEMBAHASAN

Implementasi FSM akan dilakukan di *Unity*. Setiap hewan akan dijadikan sebagai *game agent* dengan FSM agar dapat bertindak seperti hewan. Perlu diingat bahwa semua tahapan yang dilakukan adalah hanya implementasi FSM pada *game agent* saja. Jadi, untuk *gameplay* secara keseluruhan tidak dibahas disini.

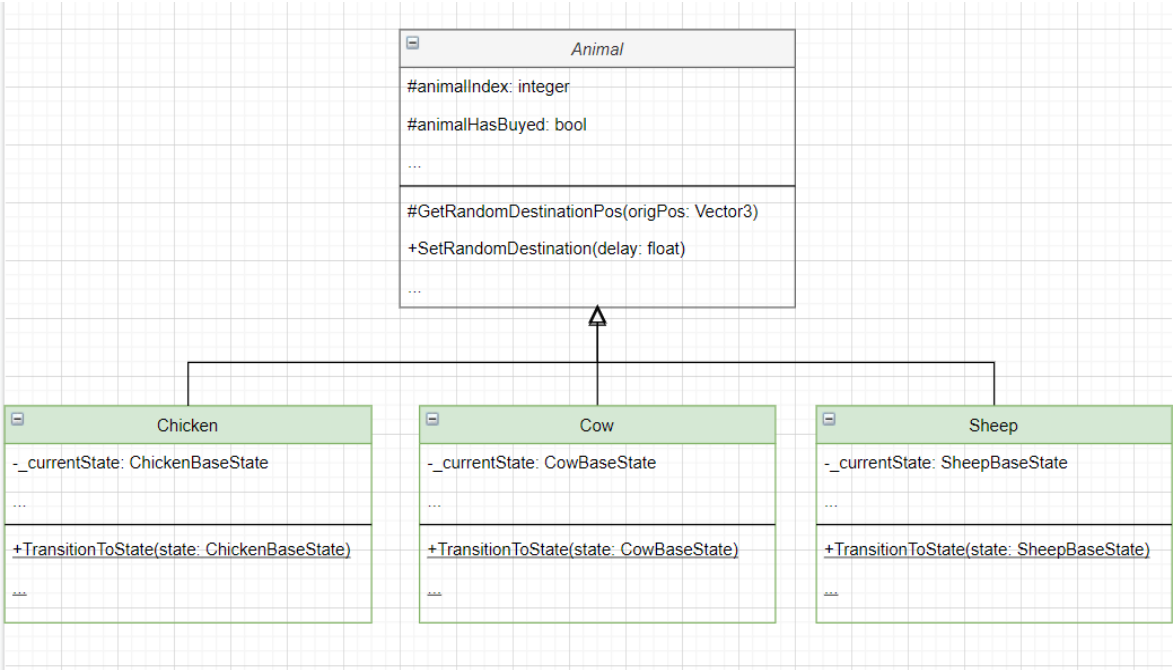
4.1. Concept

Konsep untuk FSM pada *game agent* berdasarkan *game* yang dibuat. *Game Idle B*reeder adalah *game casual top-down* bergenre *idle* dengan nuansa *relaxing*. Berdasarkan *story* dari *game*, pemain akan memerankan seorang anak bernama Arma untuk membangun ulang

peternakan kakeknya. Jadi, pemain akan membangun peternakan dari nol hingga banyak hewan. Target usia pemain yaitu 7 tahun ke atas. *Game* diusahakan dibuat sesederhana mungkin agar pemain di usia muda juga dapat memahami *game* dengan mudah. Oleh karena itu, tingkah laku pada hewan tidak perlu dibuat kompleks. Jadi, FSM yang diimplementasikan pada *game agent* cukup sederhana dengan 6 jenis *state* yang berbeda. Setiap *game agent* hewan dapat melakukan *idle*, berjalan, berlari, makan, lompat, mengepakkan sayap, dan bahkan berinteraksi dengan pemain. Namun, tidak semua hewan memiliki animasi yang sama, seperti hanya ayam yang dapat mengepakkan sayap.

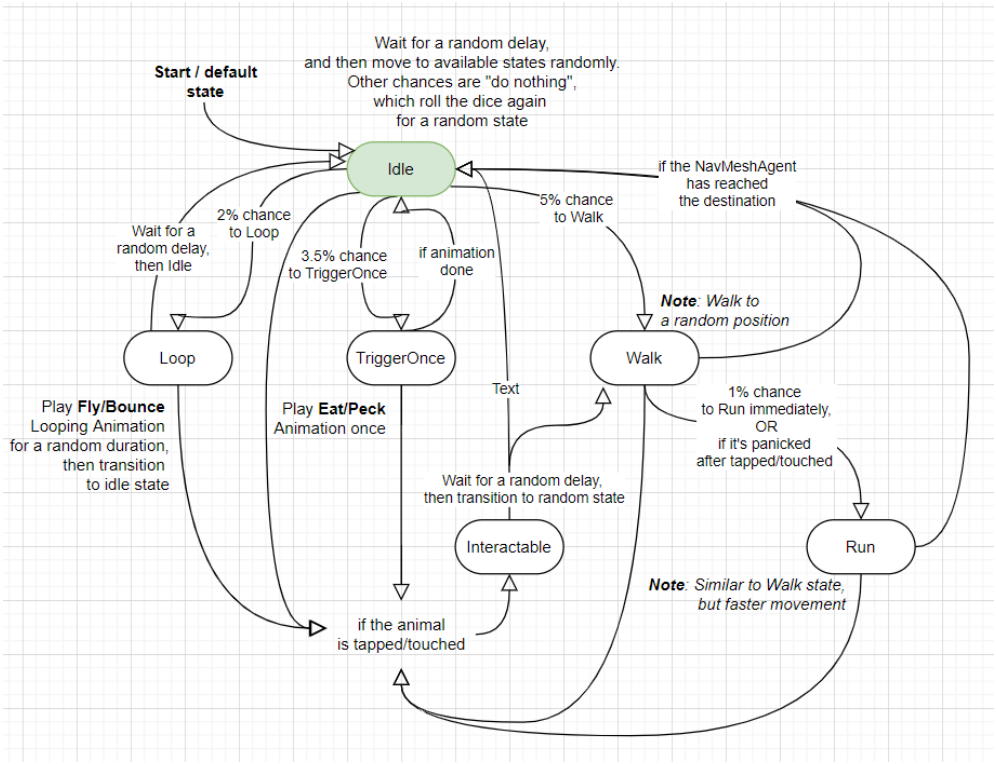
4.2. Design

Pada tahap *design*, *class* diagram dibuat untuk mendefinisikan setiap *game agent*. Setiap hewan merupakan *inheritance* dari *class* Animal, karena ada yang memiliki *variable* & *behaviour* yang mirip namun hal yang berbeda seperti animasi hewan, item yang diproduksi hewan, dll.



Gambar 2. *Class* diagram untuk *game agent Animal*

Perancangan selanjutnya yaitu merancang model FSM pada *game agent* terdapat banyak state seperti *idle, loop, trigger once, interactable, walk,* dan *run*. Semua perpindahan antar *state* adalah acak, kecuali untuk *interactable* yang dimana perlu input pemain dengan menekan hewan. Setiap perpindahan *state* selain menuju *idle* memiliki peluang yang kecil dan lebih sering *idle* atau diam di tempat agar lebih natural. Setiap hewan ditekan oleh pemain, hewan tersebut akan berpindah dengan destinasi dan kecepatan yang acak. Setiap hewan dapat memainkan animasi yang berbeda pada state *TriggerOnce* dan *Loop*. Contohnya pada state *TriggerOnce* ayam akan mematuk, sedangkan sapi memakan rumput. Dan pada state *Loop* ayam akan mengepakkan sayap, sedangkan domba akan melompat. Perbedaan antara *state TriggerOnce* dengan *Loop* adalah *state TriggerOnce* hanya dijalankan sekali pada 1 *frame*, sedangkan *Loop* dijalankan di beberapa *frame* berdasarkan waktu yang telah ditentukan.



Gambar 3. Diagram FSM *game agent*

4.3. Material Collecting

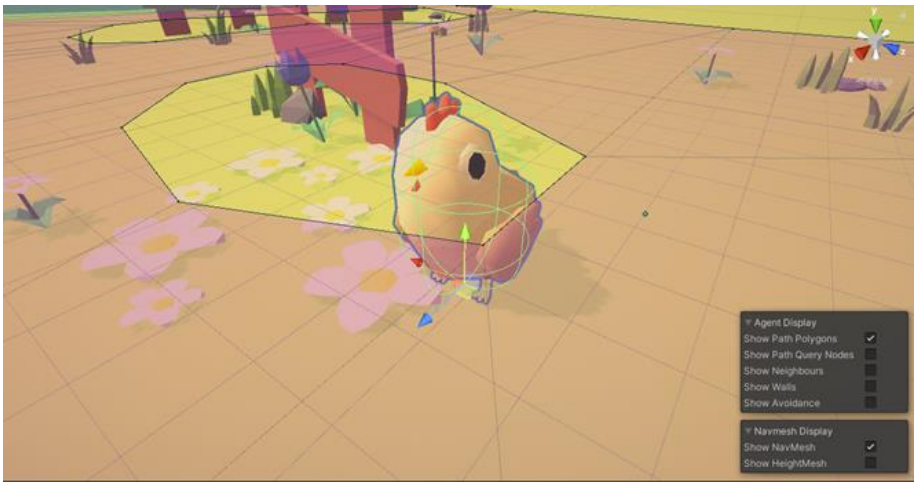
Bahan yang dikumpulkan pada tahap *material collecting* adalah asetnya. Ada banyak aset yang dibutuhkan pada *game Idle Breeder* seperti 3D model, *sound effect*, *background music*, *sprite* dan UI, dll. Aset yang digunakan untuk implementasi FSM pada *game agent* yaitu 3D model untuk tiap hewan. 3D model yang digunakan adalah aset berbayar dari *Unity Asset Store* dan sudah dibayar oleh tim *developer*. Ada banyak 3D model hewan, namun hanya 3 saja yang digunakan pada *game* yaitu ayam, sapi, dan domba.

Gambar 4. Aset 3D model hewan [22]

Gambar 5. Aset 3D model yang sudah di-*import* ke dalam Unity

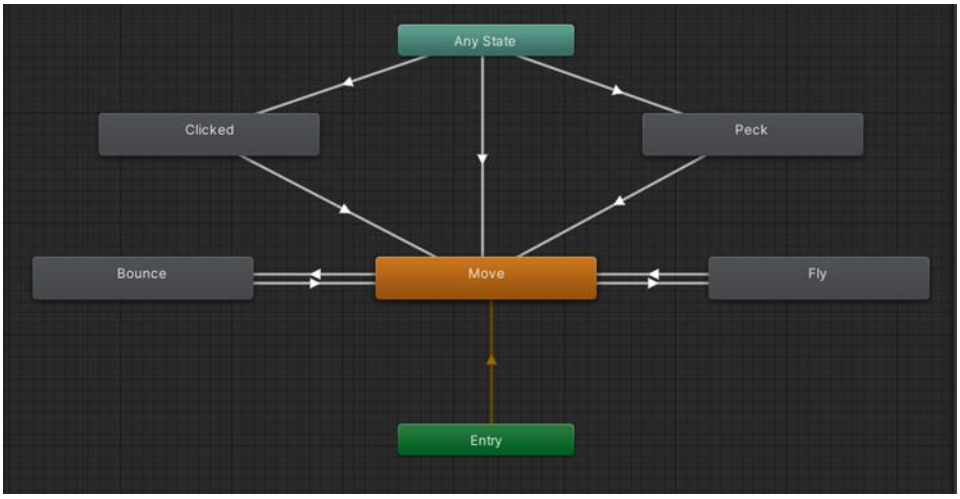
4.4. Assembly

Setelah semua bahan sudah ada, selanjutnya pada tahap *assembly* yaitu akan menyusun semuanya di aplikasi Unity seperti setup karakter *game agent*, pemrograman FSM, setup alur animasi dan transisi antar *state*, dan lain-lain. Setiap *game agent* atau hewan menggunakan *NavMeshAgent* dari Unity sebagai *pathfinding* atau navigasi agar hewan dapat bergerak dalam lingkungan 3D. Hewan dalam *game* juga memiliki *collision*-nya sendiri agar tiap hewan dapat menjaga jarak dan tidak saling tumpang tindih.

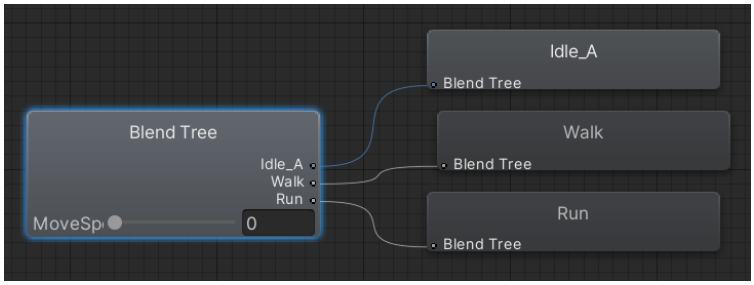


Gambar 6. Setup *game agent animal* pada *scene* Unity

Animasi pada tiap hewan menggunakan Mecanim dari Unity yang dimana juga menerapkan metode FSM. Namun perpindahan antar *state* sedikit berbeda dengan model FSM yang digunakan untuk *behaviour* hewan, karena membutuhkan kondisi tertentu agar bisa transisi antar animasi *state*. Pada *state Move* menggunakan *Blend Tree* agar transisi pergerakan hewan lebih mulus.



Gambar 7. Setup *state* animasi dan transisi *game agent animal* pada Mecanim Unity



Gambar 8. *Setup blend tree* perpindahan antar *state movement*

Setiap hewan akan diberikan *class BaseState* sebagai dasar untuk semua *state* yang akan diimplementasi. Contohnya untuk ayam ada *class ChickenBaseState* yang digunakan khusus untuk ayam. *Method EnterState* akan dijalankan sekali pada saat perpindahan dari *state* lain ke *state* tersebut. Dan *method Update* dijalankan di setiap frame. Jadi, sama seperti *method Start* dan *Update* dari *MonoBehaviour*. *Class BaseState* dijadikan sebagai *abstract* karena harus diimplementasi sebagai *state* yang spesifik seperti *idle*, *walk*, *run*, dsb.

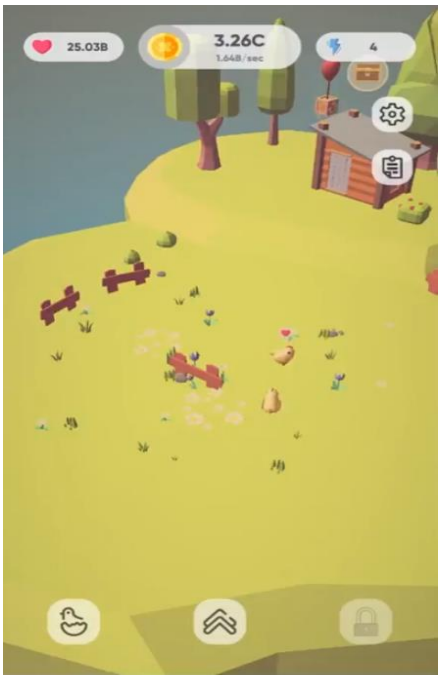
```
namespace Kigames.IdleBreeder.Gameplay
{
    9 usages  6 inheritors  Arsal  1+6 exposing APIs
    public abstract class ChickenBaseState
    {
        #region Methods

        Frequently called  1 usage  6 overrides  Arsal
        public abstract void EnterState(Chicken chicken);

        Frequently called  1 usage  6 overrides  Arsal
        public abstract void Update(Chicken chicken);

        #endregion
    }
}
```

Gambar 9. *Class ChickenBaseState* sebagai dasar *state*

Gambar 10. Daftar *state* yang sudah diimplementasi berdasarkan *ChickenBaseState*

4.5. Testing

Pada *gameplay*, pemain dapat membeli hewan untuk dapat memproduksi *item*. Contohnya ayam memproduksi telur. Setiap hewan dapat bergerak dan memainkan animasi secara acak, seperti mematuk atau mengepakkan sayap. Pemain juga dapat berinteraksi dengan hewan dengan menekan hewan itu sendiri dan *heart icon* di atasnya. *Heart* yang dimaksud adalah *currency* yang digunakan untuk *upgrade* popularitas maksimal sehingga pemain dapat menambahkan banyak hewan lagi.



Gambar 11. *Gameplay* Idle Breeder pada *game agent* ayam

Pada pengujian *black-box testing* sudah dilakukan *playtest* oleh *developer* pada masing-masing *device* android untuk tiap *state* beserta fitur lainnya. Hasil pengujian adalah berdasarkan dari pengujian untuk semua hewan yaitu ayam, sapi, dan domba.

Tabel 1. *Black-box testing* hasil implementasi FSM

No.	Nama	Luaran yang diharapkan	Hasil Uji
1	*State TriggerOnce*	Mematuk/makan	Berhasil
2	*State Loop*	Mengepakkan sayap/lompat	Berhasil

3	*State Idle*	Diam ditempat/menunggu	Berhasil
4	*State Walk*	Berjalan	Berhasil
5	*State Run*	Berlari	Berhasil
6	Dapat menghasilkan *heart*	Muncul *icon* hati	Berhasil
7	Dapat memproduksi *item*	Muncul *item* produksi	Berhasil

Pengujian selanjutnya bertujuan untuk mengevaluasi kelebihan dan kekurangan dari implementasi *Finite State Machine* (FSM) pada *game Idle Breeder*. Hasil pengujian menunjukkan beberapa perbedaan yang terjadi setelah implementasi FSM. Salah satunya adalah penggunaan *script* yang dapat digunakan kembali (*reusable*) berkat pendekatan modular, sehingga memungkinkan penggunaan *template* yang sama berkali-kali. Hal ini mengarah pada pengimplementasian kode yang lebih efisien dan mengurangi redundansi, serta menghemat waktu dalam proses implementasi. Selain itu, implementasi FSM juga menawarkan skalabilitas yang tinggi, yang memungkinkan *game* ini dapat terus dikembangkan dalam jangka waktu yang panjang dan lebih mudah untuk dilakukan *maintenance*. Meskipun performa setelah implementasi FSM sedikit lebih berat, namun perbedaannya tidak signifikan karena FSM yang diimplementasikan dalam *game* ini masih sederhana dan tidak terlalu kompleks.

4.6. Distribution

Untuk FSM yang diimplementasi pada *game agent* hewan sudah selesai dan diuji sehingga dapat diterapkan langsung pada *scene Unity*. Kemudian di-*apply* ke *branch* utama pada github. Untuk *game Idle Breeder* itu sendiri sudah dikembangkan dan dibangun menjadi format *.apk agar dapat diinstal di device android.

5. KESIMPULAN

Berdasarkan implementasi *Finite State Machine* (FSM) pada *game Idle Breeder*, dapat disimpulkan bahwa setiap hewan sudah dapat bergerak dan berinteraksi sesuai desain. Setelah diuji kelebihan dan kelemahan dari FSM bahwa *script* yang dibuat modular yang berarti *reusable* atau dapat digunakan kembali. Hal tersebut juga lebih efisien dalam kode yang diterapkan tidak *redundant* serta waktu yang digunakan untuk implementasi sedikit. Kemudian skalabilitas yang tinggi sehingga dapat dikembangkan pada jangka waktu yang panjang dan juga mudah untuk dilakukan *maintenance*. Dari segi performa setelah implementasi FSM memang akan lebih berat, namun tidak signifikan dikarenakan yang diimplementasikan tidak kompleks dan masih sederhana. Jadi, FSM hanya diimplementasikan pada game jika ingin memiliki skalabilitas yang tinggi dan memiliki sistem yang kompleks.

REFERENSI

[1] G. Tilak and T. M. Vidyapeeth, “A Study of advantages of playing video games for people,” *Pramana Res. J.,* vol. 9, no. 4, pp. 272–278, 2021.

[2] Andi et al., “Game development "kill corona virus" for education about vaccination using finite state machine and collision detection,” *J. Kinetik*, vol. 7, no. 4, pp 317-326, 2022.

[3] R. Andrea and A. Nurhuda. “Developing Edu-Game “Ulun Smart-Kid” Learning Media of Banjar Language and Game Agent with Finite State Machine Model," *International*

Journal of Education and Management Engineering (IJEME), vol. 10, no. 5, pp. 10-16, 2020, doi: 10.5815/ijeme.2020.05.02.

[4] M. Morosan, "Automating Game-design and Game-agent balancing through Computational Intelligence," Ph.D. thesis, Dept. Comput. Sci. & Elect. Eng., Univ. Essex, Essex, England, 2019. [Online]. Available: https://repository.essex.ac.uk/24233/1/THESIS.pdf.

[5] R. Andrea, S. Wijayanti, and Nursobah, "Finite State Machine Model in Jungle Adventure Game an Introduction to Survival Skills," vol. 13, no. 4, pp. 55-61, 2021.

[6] D. S. Hormansyah, A. R. T. H. Ririd, and D. T. Pribadi. "Implementasi FSM (Finite State Machine) Pada Game Perjuangan Pangeran Diponegoro," *J. Informatika Polinema*. vol. 4, no. 4. 2018.

[7] D. Jagdale, "Finite State Machine in Game Development," *IJARSCT*, vol. 10, no. 1, 2021.

[8] R. A. Elhassan, A. Yousif, and T. H. Suliman, "Entrepreneurial Development of "Ojek Sampah" (OJAH) through Android Applications," *IJIEEB*. vol. 13, no. 4, 2021.

[9] Nursobah, R. Andrea, and B. Kurniawan, "Development Finite State Machine Agent in Edugame "Hangug Word" Learning Media of Korea Hangul Letters," *J. MIB*, vol. 5, no. 2, pp. 669-675, 2021.

[10] A. F. Pukeng et al, "An intelligent agent of finite state machine in educational game Flora the Explorer," *J. Phys*, vol. 1341, no. 4, 2019.

[11] W.M. Azzakki, and D. Krisbiantoro, "Penerapan Media Pembelajaran Interaktif Pada Mata Pelajaran Sistem Pengapian Sebagai Upaya Membantu Belajar Siswa Kelas Xi Teknik Sepeda Motor (Studi Kasus: SMK Bina Mandiri)," *JOISM*, vol. 3, no. 2, 2022.

[12] Mustika, "Rancang Bangun Aplikasi Sumsel Museum Berbasis Mobile Menggunakan Metode Pengembangan Multimedia Development Life Cycle (MDLC)," *J. Mikrotik*, vol. 8, no. 1, 2018.

[13] S. R. Hernawan, "Penerapan Metode Finite State Machine Pada Game "The Mahasiswa" Guna Membangun Perilaku Non Playable Character," M. S. thesis, Dept. Industrial Tech, Univ. Islam Indo (UII), Yogyakarta, Indonesia, 2018. [Online]. Available: https://dspace.uii.ac.id/handle/123456789/12528.

[14] N. Sutikno, D. H. Setiabudi, and A. N. Tjondrowiguno, "Penerapan Finite-State Machines untuk Peningkatan Performa Frame Per Second dalam Game Multiplayer Real Time Strategy," *J. Infra*, vol. 7, no. 2, 2019.

[15] W. N. Cholifah, Yulianingsih, and S. M. Sagita, "Pengujian Black Box Testing Pada Aplikasi Action & Strategy Berbasis Android Dengan Teknologi Phonegap," *J. STRING*, vol. 3, no. 2, 2018, doi: 10.30998/string.v3i2.3048.

[16] E. W. Hidayat, A. N. Rachman, and M. F. Azim, "Penerapan Finite State Machine pada Battle Game Berbasis Augmented Reality," *J. Edukasi & Penelitian Informatika (JEPIN),* vol. 5, no. 1, 2019, doi: 10.26418/jp.v5i1.29848.

[17] H. F. Ramadhan, S. H. Sitorus, and S. Rahmayuda, "Game Edukasi Pengenalan Budaya Dan Wisata Kalimantan Barat Menggunakan Metode Finite State Machine Berbasis Android," *J. Komputer dan Aplikasi (CODING)*, vol. 7, no. 1, pp. 108-119, 2019.

[18] M. K. H. A. Baani, "Penerapan Metode Finite State Machine Pada Game Pride of Battle," *J. Mahasiswa Teknik Informatika (JATI),* vol. 3, no. 1, 2019.

[19] E. Yulsilviana, and H. Ekawati, "Penerapan Metode Finite State Machine (FSM) Pada Game Agent Legenda Anak Borneo," *J. Sebatik*, vol. 23, no. 1, pp. 116-123, 2019. Available: https://jurnal.wicida.ac.id/index.php/sebatik/article/view/453.

[20] A. Solihin, E. W. Hidayat, and A. P. Aldya, "Application of the Finite State Machine Algorithm on 2D Platformer Rabbit Games vs Zombies," *J. Online Informatika* (*JOIN)*, vol. 4, no. 1, pp. 33-38, 2019, doi: 10.15575/join.v4i1.293.
[21] D. D. S. Fatimah, E. Satria, and F. Hermawan, "Penerapan Finite State Machine pada Alpha Utopia Menggunakan Metode Game Development Life Cycle," *J. Algoritma*, vol. 20, no. 1, pp. 120-128, 2023.
[22] Omabuarts Studio. "Quirky Series - Animals Mega Pack Vol 1." https://assetstore.unity.com/packages/3d/characters/animals/quirky-series-animals-mega-pack-vol-1-137259 (accessed Jul. 18, 2023).