**Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer** e-ISSN: 2548-964X
Vol. 2, No. 10, Oktober 2018, hlm. 3618-3623 http://j-ptiik.ub.ac.id

# Penerapan Konsep *State Pattern* Pada *Game Engine* (Studi Kasus *Game Wipe It Off*)

**Firadi Surya Pramana[1], Eriq Muhammad Adams Jonemaro[2], Muhammad Aminul Akbar[3]**

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]fspramana@gmail.com, [2]eriq.adams@ub.ac.id, [3]muhammad.aminul@ub.ac.id

**Abstrak**

*Game* yang menggunakan *state machine* sebagai dasar pengembangan perilaku agen bukan merupakan hal baru lagi. Namun, masih belum banyak *game engine* yang dapat menangani kebutuhan tersebut. *State pattern* merupakan salah satu *design pattern* yang dapat menangani kebutuhan *state machine* pada *game*. Perancangan *state pattern* dilakukan untuk diterapkan dalam *game engine*. Komponen yang terdapat dalam *state pattern* adalah *Initial State*, *Check State*, dan *Handle State*. Perancangan *finite state machine* yang digunakan dalam *game* juga dilakukan. Komponen yang terdapat dalam *finite state machine* adalah *Idle State*, *Moving State*, dan *Cleaning State*. Implementasi *state pattern* pada *game engine* dilakukan dengan membuat kelas *interface* dan menjadi parent dari seluruh *state* dalam *finite state machine*. Implementasi *finite state machine* pada *game* dilakukan dengan memisahkan seluruh state menjadi kelas yang berbeda dan melakukan pewarisan sifat dari kelas *interface*. Pengujian *state pattern* pada *game engine* dilakukan dengan *white-box testing*. Didapatkan bahwa hasil pengujian dari seluruh komponen state pattern adalah valid. Pengujian *finite state machine* pada *game* dilakukan dengan *black-box testing*. Didapatkan bahwa hasil pengujian perpindahan *state* dari *finite state machine* adalah valid. Dengan keberadaan *game engine* ini diharapkan bahwa pengembang *game* tidak lagi kesulitan dalam mengembangkan *game* yang memiliki *state machine* sebagai dasar kebutuhannya.

**Kata kunci**: *Game engine, Game, State, State pattern.*

**Abstract**

*Game that uses state machine as their agent behaviour decision making is common nowaday. However, there still lots of game engine that can't handle that requirement. State pattern is one of design pattern that could handle that requirement. Components of state pattern on game engine are defined and given the name of Initial State, Check State, and Handle State. Components of finite state machine on game are defined and given the name of Idle State, Moving State, and Cleaning State. State pattern on game engine implemented as a interface class that become thhe parernt of each state on finite state machine. Finite state machine on game implemented as an individual class that inherit the interface class from state pattern. State pattern tested with white-box testing and all the compoent are given the value of valid. Finite state machine tested with black-box testing and all the transitions are succeded and given the value of valid. With this game engine, therefore there is no game developer that will experience difficulty in developing game that uses state machine as their basic need.*

**Keywords**: *Game engine, Game, State, State pattern.*

## 1. PENDAHULUAN

Dalam proses pembuatan *game*, dibutuhkan *game engine* yang berperan sebagai dasar kontrol dari *game*. G*ame engine* dibuat dan diatur hanya untuk menjalankan game di *platform* tertentu (Gregory, 2015). Saat ini, banyak *game engine* yang dikembangkan untuk memenuhi kebutuhan pengembang *game*. Namun, terdapat *game engine* yang sulit untuk dipelajari sehingga dapat memakan waktu dan tidak efisien dalam proses pengembangan *game*. Pengembangan *game engine* seharusnya tidak hanya memperhatikan performa yang lebih baik serta fungsionalitas yang kompleks saja, tetapi juga harus mempertimbangkan struktur internal yang baik (Maggiorini et al., 2016). Oleh karena

itu, diperlukan adanya *design pattern* dalam pengembangan *game engine*.

*Design pattern* digunakan untuk pengembangan *game engine* berdasarkan pola yang disediakan sehingga *game engine* dapat dikembangkan dengan sistematis. Tujuan dari *design pattern* adalah agar kode yang dibuat mudah dipahami serta mempercepat pengerjaan dari fitur yang ingin diterapkan. Penelitian Guana et al. (2015), Maggiorini et al. (2016), dan Munro et al. (2009) tentang arsitektur *game engine* diketahui bahwa *design pattern game engine* sangat berperan dalam pembangunan *game engine* yang baik dan mudah dipahami oleh pengembang *game*.

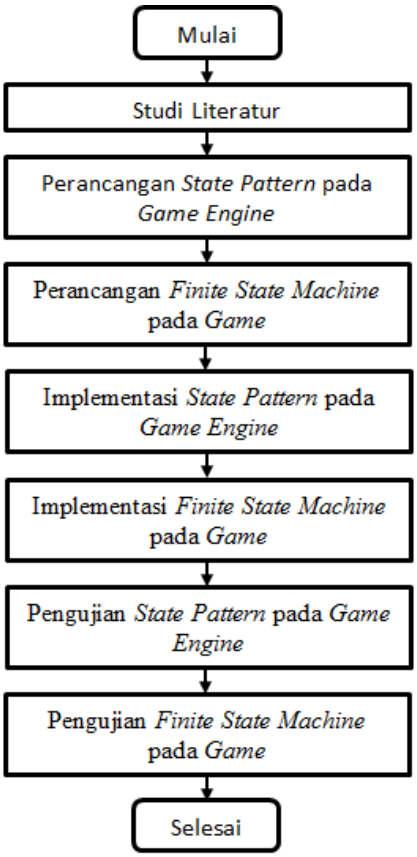
Salah satu dari *design pattern* untuk pengembangan *game engine* adalah *state pattern*. *State pattern* adalah pola dimana *game* memanfaatkan *state machine* untuk memodelkan kebutuhan *game* (Nystom, 2014). Pola ini memungkinkan *game* untuk menangani kebutuhan yang tidak disediakan oleh pengecekan kondisi secara konvensional. Dengan pola ini, tidak dibutuhkan penambahan kondisi baru apabila terdapat *state* baru yang ingin diimplementasikan.

Banyak *game* yang menerapkan model *state machine* sebagai dasar perhitungan dalam melakukan pemilihan aksi serta perilaku dari *game object*. Pada penelitian Dianty et al. (2015), *state machine* digunakan sebagai dasar skenario pengobatan yang dapat dilakukan pemain. Pemain dapat dengan mudah memahami pengetahuan dasar tentang pengobatan pertama yang ditawarkan oleh *game*. Pada penelitian Saini et al. (2011), *state machine* digunakan sebagai dasar strategi yang dapat dilakukan oleh *Artificial Intelligence(AI)* petarung. *AI* tersebut dapat meniru perilaku dari petarung manusia(pemain) melalui *state machine* yang telah dimodelkan. Dari penelitian di atas, disimpulkan bahwa model *state machine* berpengaruh penting terhadap kemudahan pemahaman dari sistem di dalam *game*.

Berdasarkan tinjauan di atas, penulis tertarik untuk melakukan penelitian mengenai penerapan konsep *state pattern* sebagai dasar pengembangan *game engine*. Diharapkan bahwa *game engine* yang dikembangkan dengan *state pattern* dapat memudahkan pengembang *game* dalam melakukan pembuatan *game*.

## 2. METODOLOGI

Metode penelitian yang digunakan adalah dengan melakukan penerapan dimana data dikumpulkan untuk. Adapun metode dalam penelitian ini terdapat pada Gambar 1.




Gambar 1. Metodologi penelitian

Pada tahap pertama dilakukan studi literatur yaitu mempelajari buku literatur yang berkaitan dengan *game engine* berupa media cetak dan media elektronik.

Tahap kedua dilakukan proses perancangan. Pada perancangan *state pattern*, dilakukan penggambaran alur kerja dari *state pattern*. Pada perancangan *finite state machine* dilakukan penggambaran *state* yang digunakan dalam *game*.

Tahap ketiga dilakukan proses implementasi. Pada implementasi *state pattern* dibuat sebuah kelas *interface* dan digunakan sebagai *parent* dari seluruh kelas *state*. Pada implementasi *finite state machine* dilakukan pembuatan kelas dari masing-masing *state* yang terdapat pada *state machine* dan melakukan pewarisan sifat dari kelas *interface* pada *state pattern*.

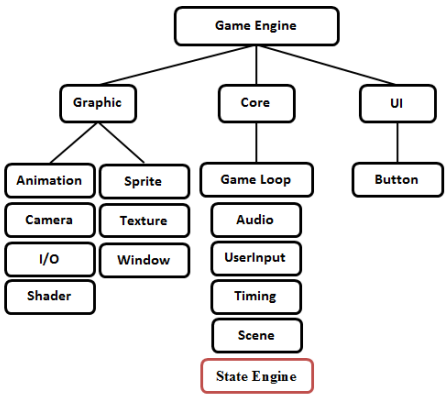
Tahap keempat dilakukan proses prngujian. Pada pengujian *state pattern* dilakukan pembuatan *pseudocode* dari alur kerja *state pattern* sehingga strukturnya dapat diuji

menggunakan pengujian *white-box*. Pada pengujian *finite state machine* dilakukan pengujian *black-box* untuk mengetahui keberhasilkan transisi antar *state*.

## 3. PERANCANGAN

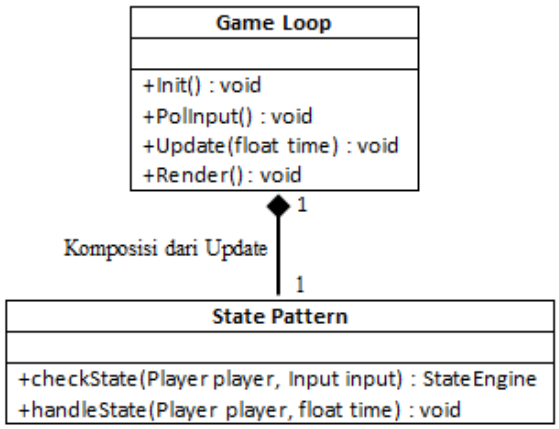
### *3.1 State Pattern*

*State pattern* digunakan untuk mengenkapsulasi setiap *state* yang berada pada *state machine*. *State pattern* berada di dalam segmen *Core* pada *game engine* karena merupakan komponen utama yang mengatur jalan dari *game*. Penempatan komponen *state pattern* terdapat pada terdapat pada Gambar 2.



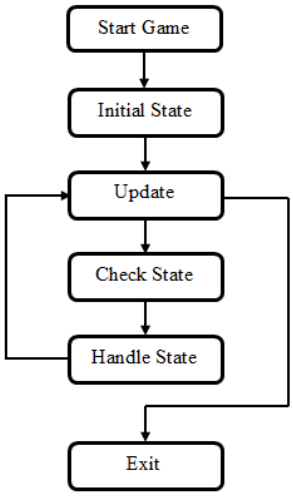

Gambar 2. Komponen *game engine*

*State pattern* dirancang dalam bentuk kelas *interface*. Kelas tersebut menjadi *parent* dari seluruh *state* dalam *state machine* pada *game*. Karena seluruh *state* memiliki perilaku yang berbeda, pembuatan fungsi di dalam kelas tersebut juga harus dapat diubah sesuai dengan perilaku *state* yang diwarisi. *State pattern* merupakan bagian dari *game loop* yang berjalan pada saat *Update game engine* dilakukan. *State pattern* tidak akan berjalan apabila *game loop* tidak berjalan. Oleh karena itu, *state pattern* memiliki hubungan komposisi dari *game loop*. Class diagram *state pattern* terdapat pada Gambar 3.




Gambar 3. *Class diagram state pattern*

Selanjutnya, perancangan alur *state pattern* dilakukan seperti yang tertera pada Gambar 4.

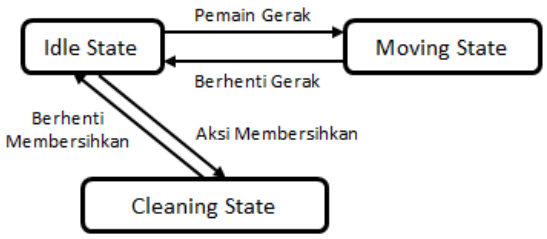



Gambar 4. *State pattern*

Terdapat tiga komponen penting yang harus dirancang. Komponen tersebut adalah *Initial State*, *Check State*, dan *Handle State*. Alur dari *state pattern* dimulai dari *intial state* pertama pada *state machine*. Pada proses tersebut dibuat sebuah variabel yang menyimpan kondisi awal dari *state*. Proses *update* adalah proses dimana game sedang berlangsung. Proses ini dilakukan dengan cara iterasi. Pada proses ini, dilakukan dua proses yaitu *check state* dan *handle state*. Proses *check state* adalah proses pengecekan apabila terdapat masukan dari pemain yang dapat mengakibatkan transisi *state* sehingga state dalam kondisi yang tepat. Proses *handle state* dilakukan untuk menjalankan seluruh *behaviour* atau perilaku dari pemain ataupun agen. Seluruh proses akan berhenti pada saat proses update berhenti dan keluar dari program.

### *3.2 Finite State Machine*

*Finite state machine* digunakan pada *game Wipe It Off* untuk mengenkapsulasi perilaku dari pemain serta bagaimana cara untuk berpindah menuju *state* lain. Finite state machine yang digunakan terdapat pada Gambar 5.



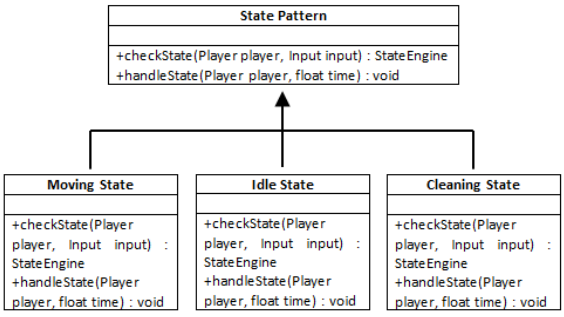

Gambar 5. *Finite state machine*

*State* pada *game* ini yaitu *Idle state*, *Moving state*, dan *Cleaning state*. Pada saat game pertama kali dimulai, pemain akan memasuki *Idle state* dimana pemain hanya diam tidak melakukan aktivitas apapun.

Pada saat terdapat aksi pemain bergerak, maka terjadi transisi dari *Idle state* menuju *Moving state*. Pada *Moving state*, pemain dapat bergerak menuju arah yang diinginkan oleh pemain namun pemain tidak dapat melakukan aksi membersihkan. Saat pemain berhenti bergerak, maka terjadi transisi dari *Moving state* menuju *Idle state*.

Pada saat terdapat aksi pemain membersihkan, maka terjadi transisi dari *Idle state* menuju *Cleaning state*. Pada *Cleaning state*, pemain tidak dapat bergerak namun pemain dapat melakukan aksi membersihkan kaca. Saat pemain berhenti membersihkan, maka terjadi transisi dari *Cleaning state* menuju *Idle state*.

Setiap kelas *state* dalam *finite state machine* merupakan anak dari kelas *interface* pada *state pattern*. *Class diagram* dari *finite state machine* terdapat pada Gambar 6.




Gambar 6. *Class diagram finite state machine*

Kelas *Idle State*, *Moving State*, dan *Cleaning State* merupakan pendefinisian kelas dalam *finite state machine*. Kelas tersebut mendapatkan pewarisan sifat dari kelas *State Pattern*. Setiap fungsi yang terdapat pada state pattern di-*override* oleh masing-masing kelas. Proses tersebut dilakukan karena perilaku Check State dan Handle State dari masing-masing kelas berbeda.

## 4. IMPLEMENTASI

### *4.1 State Pattern*

Implementasi *state pattern* dilakukan sesuai dengan alur yang didefinisikan pada Gambar 4. Proses *Initial State* dilakukan pada agen atau pemain yang menerapkan *finite state machine*. Inisialisasi dilakukan dengan pendeklarasian variabel bertipe data *StateEngine* dan diberi nilai state awal. Kelas *StateEngine* digunakan sebagai kelas *interface* yang menjadi *parent* dari kelas *state* pada *FSM*. Kode implementasi *state pattern* terdapat pada Tabel 1.

Tabel 1. Implementasi *state pattern*

| No | State Pattern |
|---|---|
| 1 | `void GameCore::Loop()` |
| 2 | `{` |
| 3 | `Init();` |
| 4 | `while (state != GameState::EXIT){` |
| 5 | `PollInput();` |
| 6 | `Update(deltaTime);` |
| 7 | `Render();` |
| 8 | `}` |
| 9 | `}` |
| 10 | |
| 11 | `class StateEngine {` |
| 12 | `public:` |
| 13 | `virtual ~StateEngine() {}` |
| 14 | `virtual StateEngine*` |
| 15 | `checkState(Player& player,` |
| 16 | `InputManager& input) = 0;` |
| 17 | `virtual void` |
| 18 | `handleState(Player& player, float` |
| 19 | `deltaTime) = 0;` |
| 20 | `};` |

Pada *start game*, fungsi *Loop* pada baris 1 akan dijalankan. *State* awal yang dijalankan petama kali akan dijalankan oleh fungsi *Init* pada baris 3. Selanjutnya, proses *loop* pada baris 4 akan dijalankan selama *game* tidak tertutup. Di dalam proses *Update* pada baris 7, terdapat proses *Check State* pada baris ke 14 yang melakukan pengecekan state saat ini. Eksekusi perilaku dari state dijalanankan dalam proses *Handle State* pada baris 17 dapat dilakukan setelah proses *Check State* selesai.

### *4.2 Finite State Machine*

Implementasi finite state machine dilakukan sesuai dengan rancangan *class diagram* pada Gambar 6. Seluruh *state* pada *finite state machine* melakukan pewarisan sifat dari kelas *interface state pattern*. Fungsi yang terdapat dalam setiap state di-*override* sesuai dengan perilaku dan kebutuhan dari masing-masing *state*.

## 5. PENGUJIAN DAN ANALISIS

### *5.1 State Pattern*

Pada subbab ini, akan dibahas mengenai hasil pengujian *white-box testing* dari *state pattern*. Pengujian ini bertujuan untuk mengetahui kebenaran struktur dari state pattern. Hasil dari pengujian terdapat pada Tabel 2.

Tabel 2. Pengujian *state pattern*

| Test Case | Expected Result | Result | Valid |
|---|---|---|---|
| Update true | Check State dan Handle State berjalan | Check State dan Handle State berjalan | Ya |
| Update false | Keluar dari program | Keluar dari program | Ya |

Seluruh test case yang didefinisikan mendapat hasil valid. Hasil tersebut menjelaskan bahwa pengujian sukses dan struktur dari state pattern sudah benar.

### *5.2 Finite State Machine*

Pada subbab ini, akan dibahas mengenai hasil pengujian *black-box testing* dari *finite state machine*. Pengujian ini bertujuan untuk mengetahui validitas dari transisi *state*. Hasil dari pengujian terdapat pada Tabel 3.

Tabel 3. Pengujian *finte state machine*

| Komp-onen | Test Case | Expected Result | Result | Valid |
|---|---|---|---|---|
| Idle State | Tidak ada tombol ditekan | Tetap di Idle State | Tetap di Idle State | Ya |
| Idle State | Tombol W ditekan | State berganti menjadi Moving State | State berganti menjadi Moving State | Ya |
| Idle State | Tombol SPASI ditekan | State berganti menjadi Cleaning State | State berganti menjadi Cleaning State | Ya |
| Moving State | Tombol W ditekan | Tetap di Moving State | Tetap di Moving State | Ya |
| Moving State | Tidak ada tombol ditekan | State berganti menjadi Idle State | State berganti menjadi Idle State | Ya |
| Cleaning State | Tombol SPASI ditekan | Tetap di Cleaning State | Tetap di Cleaning State | Ya |
| Cleaning State | Tidak ada tombol ditekan | State berganti menjadi Idle State | State berganti menjadi Idle State | Ya |

Seluruh test case yang didefinisikan mendapat hasil valid. Hasil tersebut menjelaskan bahwa pengujian sukses dan seluruh transisi dalam finite state machine telah benar.

## 6. KESIMPULAN

Dalam penelitian ini dapat diambil kesimpulan bahwa:

- Penerapan konsep *state pattern* pada *game engine* dapat dilakukan dengan membuat kelas *interface* yang berisi fungsi *Check State* dan *Handle State*. Kelas tersebut menjadi *parent* dari seluruh *state* dalam *finite state machine* yang akan dikembangkan dalam *game*.

- Penerapan *finite state machine* pada *game* dilakukan dengan mengonversi setiap *state* menjadi kelas yang mewarisi kelas *interface* yang telah dibuat pada *game engine*. Seluruh fungsi yang diwariskan dilakukan *overriding* sehingga setiap *state* dapat bekerja sesuai dengan kebutuhan dari *state machine*. Fungsi *Check State* diisi dengan syarat transisi dari setiap *state*. Sedangkan fungsi *Handle State* diisi dengan *behavior* dari setiap *state*.

- Hasil pengujian *state pattern* pada *game engine* dan *finite state machine* pada *game* adalah menunjukkan nilai valid. Nilai tersebut menudeskripsikan bahwa komponen telah sukses diuji dan sesuai.

Game engine dapat dikembangkan lebih lanjut lagi dengan cara optimasi komponen

dengan melakukan pengujian performa dari setiap komponen sehingga struktur dari komponen menjadi lebih baik, lebih rapi, dan lebih sederhana.

## 7. DAFTAR PUSTAKA


Ali, Z., Usman, M., 2016. A Framework for Game Engine Selection for Gamification and Serious Games. Future Technologies Conference(FTC).

C++ Language, 1985. Tersedia di:< http://www.cplusplus.com> [Diakses 16 Agustus 2017]

Dianty, E.D., Azhari, A.M., Hakim, M.F.A., Kuswardayan, I., Yuniarti, A., Herumurti, D., 2015. First Aid Simulation Game with Finite State Machine Model. International Conference on Information, Communication Technology and System(ICTS).

Gregory, Jason, 2015. Game Engine Architecture. 2nd ed. Boca Raton: CRC Press.

Guana, V., Stroulia, E., Nguyen, V., 2015. Building a Game Engine: A Tale of Modern Model-Driven Engineering. IEEE/ACM 4th International Workshop on Games and Software Engineering.

irrKlang, 2006. Tersedia di:< https://www.ambiera.com/irrklang> [Diakses 16 Agustus 2017]

Maggiorini, D., Ripamonti, L.A., Zanon, E., Bujari, A., Palazzi, C.E., 2016. SMASH: a Distributed Game Engine Architecture. IEEE Symposium on Computers and Communication(ISCC).

Munro, J., Boldyreff, C., Capiluppi, A., 2009. Architectural Studies of Game Engines - the Quake Series. International IEEE Consumer Electronics Society's Games Innovations Conference.

Nystrom, Bob, 2014. Game Programming Patterns. Tersedia di:<http://gameprogrammingpatterns.com> [Diakses 16 Agustus 2017]

OpenGL, 1992. Tersedia di:<https://www.opengl.org> [Diakses 16 Agustus 2017]

OpenGL Mathematics, 2005. Tersedia di:<https://glm.g-truc.net> [Diakses 16 Agustus 2017]

Saini, S., Chung, P.W.H., Dawson, C.W., 2011. Mimicking Human Strategies in Fighting Games using a Data Driven Finite State Machine. 6th IEEE Joint International Information Technology and Artificial Intelligence Conference.

Simple DirectMedia Layer, 1998. Tersedia di:<https://www.libsdl.org> [Diakses 16 Agustus 2017]

The OpenGL Extension Wrangler Library, 2002. Tersedia di:< http://glew.sourceforge.net> [Diakses 16 Agustus 2017]

What is a Game Engine?, GameCareerGuide, 2008. Tersedia di:<https://www.gamecareerguide.com/features/529/what_is_a_game_.php> [Diakses 16 Agustus 2017]

Wright, David R., 2005. Finite State Machine. CSC215 Class Notes. David R. Wright website, N. Carolina State Univ.

Yan-hui, W., Xia-xia, Y., He-Jin, 2011. Design and Implementation of the Game Engine Based on Android Platform. International Conference on Internet Technology and Applications(iTAP).