UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE CIENCIAS

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

Teoría de autómatas en el diseño de juegos

PROYECTO DE TESIS 1

Autor: Piero Alexis Violeta Estrella

Asesor: Victor Andrés Melchor Espinoza

Diciembre, 2021

Resumen

Esta tesis realizará una descripción del uso de la teoría de autómatas en especifica aplicación al diseño de juegos, GD, por sus siglas en ingles. Además de realizar una comparación de diversos autómatas aplicados a GD, basándose en la literatura existente. Finalmente se pretende servir como marco teórico para futuros proyectos e implementaciones de videojuegos.

Índice general

Índice de figuras

Índice de Acrónimos

GDD	Game Design Document
GD	Game Design
ETC	Etcétera
AFD	Autómata Finito Determinista
AFND	Autómata Finito No Determinista

Agradecimientos

Principalmente a mis padres por toda la confianza y paciencia depositada en mi durante mi formación académica.

A mis amigos por su apoyo emocional brindado en momentos críticos.

A mis compañeros de carrera por haber podido adquirir conocimiento juntos durante estos años.

Finalmente agradezco a mi asesor de tesis MSc. Victor Melchor, ya que el curso de Teoría de Autómatas, Lenguajes y Computación dictado por su persona fue crítico para el interés en la rama y elección del tema.

Capítulo 1

Introducción

1.1. Motivación

Desde sus inicios, los videojuegos han sido una forma de entretenimiento que no pasaba desapercibida. Desde grandes locales donde la gente se reunía a jugar los famosos juegos Arcade, hasta los videojuegos casuales de teléfono móvil que tenemos hoy en día.

Siendo una industria en constante crecimiento, durante los últimos años, la industria de los videojuegos ha tenido un crecimiento gigantesco a comparación a otras industrias, llegando a superar incluso a industrias que tienen más años en el mercado como el cine o los deportes. Esto debido a la gran concentración de trabajo que esta industria mueve, ya que no solo están involucrados aquellos responsables de crear el videojuego, sino también otros sectores no tan evidentes que tienen un involucramiento indirecto con el videojuego en cuestión. Entre ellos podemos mencionar a: músicos, escritores, artistas, abogados, contadores, etc.

Sin embargo, el primer paso de toda está gran industria es siempre el mismo: crear un videojuego. Como toda industria, esta también esta en constante evolución por ende la creación de un videojuego también lo está. Durante muchos años, la forma en la que una persona o un grupo de personas desarrolla un videojuego ha ido cambiando. Un ejemplo concreto de este cambio puede ser la comparación en la creación de Tetris(1984), el cual fue desarrollado por una sola persona, con la creación de Half Life(1998), el cual requirió un estudio completo para su realización. En esta comparación cabe resaltar un problema

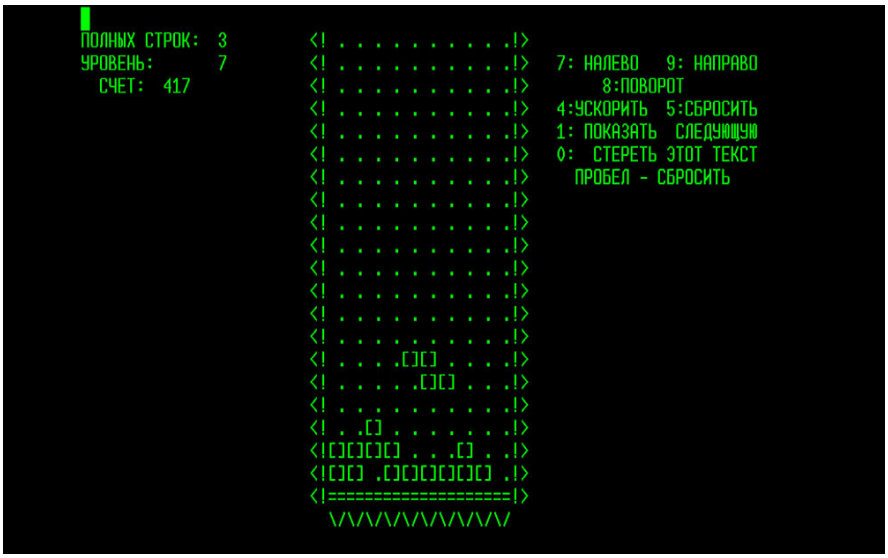

FIGURA 1.1: Juego desarrollado por una sola persona Tetris(1984)

que tuvo que afrontar Half Life a diferencia de Tetris, el cual fue la coordinación entre diferentes áreas de desarrollo. En especifico, Tetris al haber sido realizado por una persona, no tuvo que afrontar dichos problemas de coordinación y/o entendimiento.

En términos generales, dos pilares fundamentales en la creación de un videojuego son los siguientes: Diseño de juego y Programación.

El diseño de juegos es una parte fundamental en el desarrollo de videojuegos, ya que es la que se encarga de diseñar las reglas del juego y otros elementos. Por otro lado otra pieza fundamental es la programación, ya que es la que se encarga de implementar y hacer realidad las reglas anteriormente definidas por los diseñadores de juegos.

Por el ejemplo anterior mostrado, es que es necesario un buen entendimiento entre diseñadores de juegos y programadores. Un documento estándar para el proceso de desarrollo que ayuda en este entendimiento es el documento de diseño de juegos, GDD por sus siglas en ingles. Sin embargo, a medida que se hace más complejo un juego, un GDD ya no es una solución factible para este entendimiento, puesto que la cantidad de reglas definidas por los diseñadores aumenta y por tanto, la dificultad de entendimiento por parte de los programadores a los diseñadores de juegos, así como la necesidad de algún documento de menor dificultad, se incrementa.

Debido a esto, se han planteado diferentes soluciones para este problema tales como: metodologías ágiles o reuniones más frecuentes entre miembros del equipo de desarrollo. En la presente tesis se realizará una revisión a una solución diferente, la cual toma la teoría de autómatas para formalizar las reglas definidas por los diseñadores de juegos con el fin de facilitar el entendimiento por parte de los programadores.

1.2. Objetivos

1.2.1. Objetivo General

Generar un marco teórico sobre el uso de la teoría de autómatas en el diseño de juegos.

1.2.2. Objetivos Específicos

- Determinar que autómata es el indicado para cada género de juego.
- Realizar un ejemplo de diseño basado en algún videojuego ya existente.
- Analizar si el uso de autómatas en el diseño de videojuegos es beneficioso para el entendimiento programador-diseñador.

1.3. Estructura del Seminario

- **Introducción:**
  En este capítulo se introduce al lector en el tema a tratar, explicando las motivaciones y objetivos del proyecto con intención de sembrar interés por el tema.
- **Planteamiento del Problema:**
  En este capitulo se dará una descripción detallada sobre lo que consiste el problema a tratar en la presente tesis, así como la importancia por la cual esta siendo realizada el presente estudio.

- **Marco Teórico**

  En este capitulo se desarrollan conceptos teóricos previos, así como contexto histórico, investigaciones pasadas relacionadas al tema y finalmente, se concluye con una hipótesis a ser tratada en el siguiente capitulo.

- **Metodología y desarrollo**

  En este capitulo se desarrolla el objetivo de la presente tesis, abordando el problema con las soluciones de distintos autores y brindando un análisis adicional.

- **Conclusiones y Trabajo a Futuro:**

  En este capítulo se exponen las conclusiones obtenidas de este trabajo. Adicionalmente, se proponen trabajos a futuro para la implementación de la metodología propuesta en el presente trabajo.

Capítulo 2

Planteamiento del Problema

2.1. Descripción del problema

Tal como se abordó en la introducción, el problema radica en el entendimiento por parte del programador hacia las reglas establecidas por el diseñador de juegos. Ilustraremos esta problemática con un ejemplo.

En la figura 2.1 podemos ver un juego de reglas bastantes simples, puesto que las únicas acciones reglas definidas por el diseñador de juegos son:

- mover de arriba hacia abajo a cada jugador
- el movimiento de la pelota basado en rebote

En este ejemplo es sencillo visualizar que el programador puede tener un claro entendimiento de las reglas a implementar.

Por otro lado en la figura 2.2 podemos ver un juego con mayor cantidad de reglas y acciones a implementar. Si bien el diseñador de juegos plantea todas las reglas tales como: atacar, defenderse, intercambiar de arma, caminar, correr, saltar, etc, los cuales a su vez están definidos por otras subreglas. Por ejemplo, la mecánica de atacar puede ser realizada de muchas formas, con diferentes armas, y teniendo diferentes retroalimentaciones. En un juego de esta categoría muchas veces se puede superar fácilmente varios miles reglas/subreglas definidas.

Con los ejemplos anterior mostrados, podemos ver claramente un contraste entre un juego de reglas simple y un juego con reglas complejas. Como se mencionó en la introducción, un videojuego puede ser dividido en dos

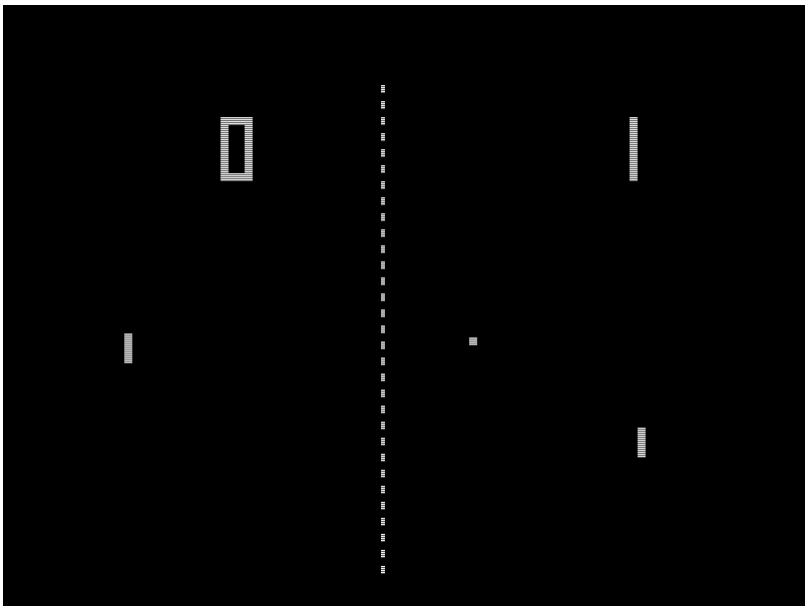

FIGURA 2.1: Juego de reglas simples: Pong(1972)

FIGURA 2.2: Juego de reglas complejas: God Of War(2018)

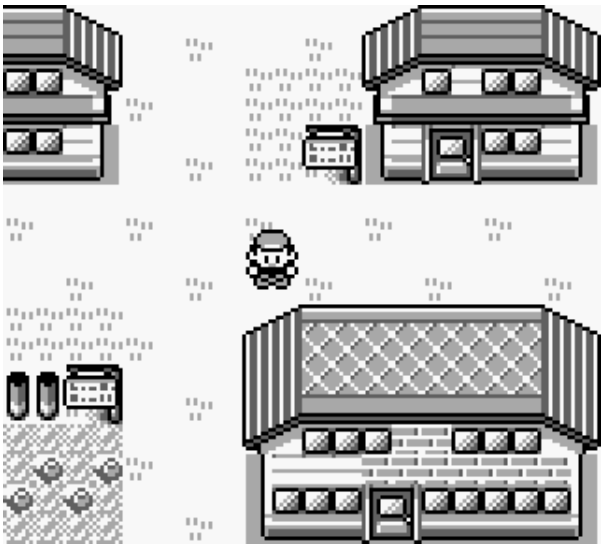

FIGURA 2.3: Juego desarrollado en lenguaje ensamblador: Pokemon Red(1996)

fase principales diseño de juegos y programación. Las reglas mencionadas anteriormente, tanto en el caso de Pong [2.1] como en el caso de God Of War [2.2], pertenecerían exclusivamente a la parte del diseño de juegos. Una vez definida dichas reglas se procede al siguiente paso aún más importante: la implementación.

La implementación de un videojuego ha variado mucho conforme el paso de los años, desde juegos que fueron realizados en lenguaje ensamblador véase el caso de Pokemon Red [2.3]; hasta un ejemplo contemporáneo como es el caso del videojuego peruano Tunche [2.4], el cual fue desarrollado en el motor gráfico Unity haciendo uso del lenguaje de programación C#.

Ambos ejemplos, a pesar de la distinta implementación que tuvieron y distancia en años de lanzamiento, probablemente enfrentaron la problemática de plasmar fielmente lo que el diseñador de juegos concibió como idea original. Más allá de superar el hecho de tener un prototipo funcional y jugable, el programador debe plasmar todas las reglas definidas por el diseñador de juegos. Lo cual no supondría mayor dificultad si habláramos de 20 o 30 reglas por implementar. Sin embargo, como se mencionó anteriormente, en juegos de estos calibres fácilmente se pueden superar los varios miles de reglas a implementar.

FIGURA 2.4: Juego desarrollado en motor gráfico Unity y lenguaje de programación C#: Tunche(2021)

En este escenario, es que es necesario tener una buena metodología de trabajo que comunique a los programadores con diseñadores de juegos, para mejorar el entendimiento para la implementación de las reglas definidas por estos últimos. Ya que se puede inferir que a mayor cantidad de reglas definidas, mayor será la dificultad de entendimiento por parte de los programadores hacia los diseñadores de juegos.

2.2. Formulación del problema

Como se menciono en la introducción, una documento que puede ayudar en el problema anterior mencionado es el GDD. Sin embargo, por lo general para una gran cantidad de reglas el documento resultaría bastante extenso.

La dificultad de entender un GDD muy complejo, para mecánicas o reglas con muchas restricciones, supone un reto para los programadores al momento de implementar el videojuego.

Como también se mencionó, existen otras metodologías además de los GDD que también se han utilizado para la explicación de reglas para facilitar el entendimiento al implementar el videojuego. Entre estas soluciones se encuentra la formalización de reglas y/o mecánicas de videojuegos haciendo

uso la teoría de la computación. En especifico, los siguientes artículos muestran que es posible tratar esta problemática haciendo uso de la teoría de autómatas.

El articulo escrito por Mushtaq[1], da una propuesta de solución al problema mencionado anteriormente. Más a detalle, se introduce la teoría de autómatas como una manera de formalizar las reglas definidas por el diseñador de juegos. Además se analiza que facilita el entendimiento para su posterior implementación por parte del programador.

Por otro lado, Abbas[2] realiza una implementación de un juego de montaña rusa usando el diseño de juegos basado en autómatas mostrando así, que es posible la implementación basada en diseño de juegos usando teoría de autómatas.

Lo mencionado nos permite ver que es posible tratar el problema planteado haciendo uso de la teoría de autómatas.

2.3. Importancia y Justificación del estudio

Ambos artículos mencionados en el apartado anterior, nos muestran la importancia del estudio de este tópico. Esta se basa, principalmente, en que dar una revisión de las soluciones del problema de entendimiento entre programadores y diseñadores de juegos, permitirá tener una base teórica solida para futuras implementaciones evitando esta problemática. Más en especifico, mejorando el flujo de trabajo al momento de programar el videojuego.

Así como también la comparación entre los diversos autómatas utilizados en el diseño de juegos, nos permitirá clasificar los diversos tipos de autómatas con el fin de saber que autómata es el mejor para la formalización de reglas en cada género de juego, para posteriormente realizar el análisis respectivo.

Capítulo 3

Marco teórico

3.1. Conceptos previos

3.1.1. Autómata finito

Hopcroft[5] nos define informalmente un autómata finito siguiendo la analogía de situaciones y protocolos. Para cada situación dada, tenemos un protocolo a seguir. La definición formal no dista mucho del ejemplo dado, se define como sigue:

Autómata Finito Determinista

Un Autómata Finito Determinista(AFD) A es una 5-upla $(Q, \Sigma, \delta, q_0, F)$, donde:

- Q denota un conjunto finito de estados.
- Σ denota el alfabeto de entrada.
- δ denota una función de transición, que toma como entrada un estado y un carácter de entrada, y retorna un estado. Formalmente: $\delta : Q \times \Sigma \to Q$
- q_0 denota el estado inicial, donde $q_0 \in Q$
- F denota un conjunto finito de estados de aceptación.

Autómata Finito No Determinista

Un Autómata Finito No Determinista(AFND) B es una 5-upla $(Q, \Sigma, \delta, q_0, F)$, donde:

- Q denota un conjunto finito de estados.
- Σ denota el alfabeto de entrada.
- δ denota una función de transición, que toma como entrada un estado y un carácter de entrada, y retorna un subconjunto de Q. Formalmente: $\delta\colon Q \times \Sigma \to \mathcal{P}(Q)$
- q_0 denota el estado inicial, donde $q_0 \in Q$
- F denota un conjunto finito de estados de aceptación.

Se puede notar que la diferencia principal entre un AFD y un AFND es la función de transición. Se suele ampliar la definición de un AFND, ampliando la definición de la función de transición como sigue: $\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$. Esto permite tener transiciones de un estado a otro sin la necesidad de un carácter del alfabeto Σ.

Para el desarrollo de esta tesis también es necesario un tipo especial de autómata finito denominado autómata de pila. Hopcroft[5] define informalmente a este autómata como un autómata finito no determinista con transiciones ε con la capacidad de almacenar caracteres en una estructura de dato tipo pila. Estos caracteres están definidos en un alfabeto adicional, asi como el tamaño de esta pila puede ser infinito. Formalmente:

Autómata de pila

Un Autómata de pila C es una 7-upla $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, donde:

- Q denota un conjunto finito de estados.
- Σ denota el alfabeto de entrada.

- Γ denota el alfabeto de pila.
- δ denota una función de transición, que toma como entrada un estado, un carácter de entrada(incluso cadena la vacía ε) y un carácter de la pila. Este a su vez retorna un subconjunto de $Q \times \Gamma^*$.
  Formalmente: $\delta \colon Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma^*)$
- q_0 denota el estado inicial, donde $q_0 \in Q$
- Z_0 denota el carácter inicial de la pila, donde $Z_0 \in \Gamma$
- F denota un conjunto finito de estados de aceptación.

Máquina de Mealy

Una máquina de Mealy D es una 6-upla $(Q, q_0, \Sigma, \Lambda, T, G)$, donde:

- Q denota un conjunto finito de estados.
- q_0 denota el estado inicial, donde $q_0 \in Q$.
- Σ denota el alfabeto de entrada.
- Λ denota el alfabeto de salida.
- T denota una función de transición, que toma como entrada un estado y un carácter de entrada. Este a su vez retorna un estado de Q.
  Formalmente: $T \colon S \times \Sigma \to S$
- G denota una función de salida que toma como entrada un estado y un carácter de entrada. Este a su vez retorna un carácter de Λ.
  Formalmente: $G \colon S \times \Sigma \to \Lambda$

3.1.2. Diseño de juegos

Schell[6] define el diseño de juegos como sigue: "El diseño de juegos es el acto de decidir lo que un juego debe ser". Esta definición viene acompañada con una aclaración que nos dice que el diseño de juegos implica hacer cientos

de decisiones, usualmente miles. Además cabe resaltar la separación que se hace del diseño de juegos con la programación de videojuegos, siendo estas dos disciplinas distintas.

El diseñador de juegos usualmente tiene a cargo múltiples tareas, solo por listar algunas tenemos:

- Experiencia
- Elementos
- Diseño de nivel
- Control de iteraciones
- Mecánicas
- Curvas de nivel

Schell[6] menciona que entre las tareas fundamentales se encuentran: Estética, historia, mecánicas y tecnología. Siendo listadas todas estas en orden decreciente respecto a cuan visual es esto para el jugador. Por ejemplo un jugador notará más fácilmente los dibujos o animaciones en un juego, que notar en que motor gráfico y lenguaje de programación fue implementado.

Podemos notar que un diseñador de juegos puede estar involucrado en muchas tareas. Sin embargo, para el fin de esta tesis, es suficiente con enfocarnos en la penúltima de las tareas listadas: Mecánicas.

Mecánicas de juego

Son las reglas y procesos que ocurren dentro un juego. Entre estas están incluidas la meta o fin a alcanzar dentro de un juego, las acciones que están y no están permitidas por los jugadores, así como que es lo que sucede después de haberse realizado una acción en especifico. Como se menciono anteriormente en juegos de gran magnitud se puede alcanzar una gran cantidad de reglas/mecánicas definidas por el diseñador de juegos.

3.2. Marco Histórico

La teoría de la computación es una rama de las ciencias de la computación, la cual fue creada en el siglo XX. En sus inicios era considerada una disciplina teórica sin mucha aplicación a problemas reales, ligada más a la matemática pura. Conforme la computación y la tecnología fue evolucionando, muchos de los modelos netamente matemáticos pudieron ser implementados. Las sub-ramas principales de la teoría de la computación son: Teoría de autómatas, teoría de la computabilidad y la teoría de la complejidad.

En particular, como se mencionó, la teoría de autómatas es una sub-rama de la teoría de la computación, la cual ha tenido un impacto y aplicación importante en muchos ámbitos. Teniendo múltiples aplicaciones tales como: Diseño de hardware, compiladores, e inteligencia artificial. Esta se centra en el estudio de un *Autómata*, el cual es un concepto matemático que permite el modelamiento de muchos problemas. Así mismo, existen múltiples tipos de autómatas, como por ejemplo la maquina de Turing, en la cual estuvieron basados los primeros computadores programables.

Coincidentemente, los videojuegos también tuvieron su surgimiento en el siglo XX. Comenzando como un proyecto científico el cual involucraba Tubos de Rayos Catódicos, hasta la gran industria que es actualmente. Conforme avanzaron los años, mayores puestos de trabajos fueron generados por los videojuegos, y así mismo, se exploraron nuevas formas de desarrollar un videojuego. Adaptando el estudio que se realizaba para el diseño de juegos de mesas, se pudo dar el surgimiento al diseño de juegos moderno.

Es en este contexto que se intenta dar una intersección entre el diseño de juegos y la teoría de autómatas.

3.3. Investigaciones relacionadas al tema

A continuación se mostrará algunos artículos relacionados que aportaron al desarrollo de esta tesis:

3.3.1. Computing Game Design with Automata Theory[1]

Resumen

En este articulo se discute de manera general la formalización de un videojuego a reglas matemáticas como funciones y autómatas finitos, se aborda la relación entre el diseño de juegos y la teoría de autómatas haciendo énfasis en la poca extensión que este tópico ha tenido en su estudio. Además se hace el diseño de 5 niveles de un juego arcade haciendo uso de un Autómata Finito Determinista y un Autómata Finito No Determinista.

Objetivos

Realizar el diseño de un videojuego arcade haciendo uso de la teoría de autómatas.

Conclusiones

Se llega a la conclusión de que realizar el diseño de un videojuego haciendo uso de la teoría de autómatas, así mismo se menciona que se puede utilizar otros tipos de autómatas distintos a los usados para diferente escenarios. Se realiza un pequeño análisis del porque es preferible utilizar un AFD que un AFND en la implementación.

3.3.2. A Roller Coaster Game Design using Automata Theory[2]

Resumen

En este articulo se realiza el diseño de de un juego de montaña rusa haciendo uso de la teoría de autómatas. Se menciona que el diseño de juegos es un área donde puede ser aplicada satisfactoriamente la teoría de autómatas. Además realizan un primero el diseño preliminar utilizando un AFND, para posteriormente convertirlo a un AFD, el cual sera utilizado para el desarrollo del código fuente.

Objetivos

Mostrar el proceso de diseño de un videojuego de montaña rusa, haciendo uso preliminar de un AFND.

Conclusiones

Se logró el diseño del juego planteado haciendo uso de herramientas de la teoría de autómatas. Además se notó que una vez realizado el diseño con esa metodología se tuvo una menor cantidad de errores(bugs), al momento del desarrollo, en comparación a otra herramienta de diseño de juegos. Se dejó a futuros estudios la posibilidad de sustituir técnicas de ingeniería de software como los diagramas de caso de uso, por técnicas basadas en teoría de autómatas. Así también el futuro estudio del uso de expresiones regulares para poder determinar todos los escenarios posibles del juego.

3.3.3. An Infinite Runner Game Design using Automata Theory[3]

Resumen

Muestra el diseño de un juego de tipo runner infinito similar al popular juego Flappy Bird, haciendo uso de un autómata diferente a AFD o AFND. Se utilizó una Máquina de Mealy para el diseño del juego, además se observó que es más entendible para el programador así como también se puede extender a automatización mas avanzada.

Objetivos

Realizar el diseño de juegos utilizando teoría de autómatas, utilizando un autómata conocido como Máquina de Mealy.

Conclusiones

Se logró el diseño de un juego infinito de tipo Runner. Observando así que en la fase de implementación, menor cantidad de errores fueron encontrados a comparación de otras técnicas. Además que se observó que todas las entradas invalidas fueron abarcadas en la etapa de diseño y no en la etapa de implementación como es usual. Se dejó a futuros estudios la posibilidad de utilizar otras herramientas de computación en el diseño de juegos, tales como: inteligencia artificial o teoría de la complejidad.

3.3.4. Re-designing the Pacman Game using Push Down Automata [4]

Resumen

Realiza un rediseño del clásico videojuego Pacman para estar completamente basado en el diseño de juegos basado en teoría de autómatas. Se introduce un nuevo autómata el cual es el autómata con pila. Esto con el fin de encontrar en el diseño de juegos basado en autómatas una alternativa a los principios clásicos de la ingenieria de software. Además se desarrollo una implementación en el motor gráfico Godot, para finalmente realizar pruebas para comprobar si las salidas producidas por el autómata de pila es el mismo que el del juego original.

Objetivos

Realizar el rediseño del videojuego Pacman utilizando diseño de juegos basado en autómatas, en especifico, un autómata de pila.

Conclusiones

Se mostró que hay consistencia en la salida producida por el autómata de pila y las salidas esperadas. Además se menciona que el uso de un autómata de pila para el diseño de un juego es una buena practica de diseño para aquellos

que no han tenido experiencia previa con la programación. Sin embargo, también ayuda a que la implementación este mas organizada. También se nota que esta propuesta es una buena alternativa de reemplazo a los clásicos diagramas de caso de uso.

3.4. Hipótesis

Habiendo mencionado los conceptos previos, objetivos y planteamiento del problema. Procedemos a plantear la hipótesis en la que girará en torno esta tesis.

Hipótesis general

Es posible generar un marco teórico sobre el diseño de juegos basado en autómatas.

Hipótesis Especificas

- La teoría de autómatas trae un beneficio al entendimiento por parte del programador de reglas y/o mecánicas diseñadas por el diseñador de juegos.
- El uso de un tipo de autómata adecuado para determinados tipos de videojuegos puede facilitar su implementación en gran medida.

Capítulo 4

Metodología y desarrollo

4.1. Metodología

Con el fin de obtener un marco teórico y poder obtener una clasificación autómata - tipo de juego. Se dividirá este sección en 3 secciones: Selección de una instancia de juego a analizar, diseño usando teoría de autómatas y análisis.

4.1.1. Selección de una instancia de juego

En esta parte se escogerá solo una parte de un juego determinado, tratando de escoger una parte que abarque la mayor cantidad de aspectos comunes del juego. Por ejemplo, si es un juego que consta de varios niveles, se escogerá solo un nivel a modelar. Se realizara esto para múltiples juegos. Para facilitar esto ultimo se utilizaran los juegos analizados en los cuatro artículos listados en investigaciones relacionadas al tema.

4.1.2. Diseño usando teoría de autómatas

Esta parte consistirá en el modelamiento del diseño de juego de la instancia de juego escogida haciendo uso de la teoría de autómatas. Se presentará un grafo que represente al autómata visualmente, asi como la explicación de cada estado.

4.1.3. Análisis

En esta parte se dará argumentos de porque cierto tipo de autómata es bueno en el diseño del juego escogido, y un análisis de si es posible extender este tipo de autómata al genero que pertenece el juego analizado en el apartado anterior.

4.2. Desarrollo

4.2.1. Juego Arcade

Selección de una instancia de juego

El juego que analizaremos es el juego arcade propuesto por Mushtaq[1]. Es un juego que consiste en mover al jugador con un arma asignada a través de un escenario (nivel), atacando y evitando ser dañado por enemigos. Cada nivel consta de dos tipos de nivel *good* y *danger* Para la instancia de este juego se escogerá el nivel 2. Este nivel dota al jugador de un arma de tipo *Machine Gun* como arma por defecto y tiene los siguientes enemigos:

- 12 soldados con el arma de tipo *Uzi*
- 3 enemigos con el arma de tipo *Machine Gun*.
- 10 tanques con arma de tipo *Cannon*
- 2 enemigos con 5 veces el daño de los anteriores enemigos al final del nivel.

A continuación se lista los alfabetos de entrada que usara el autómata que se construirá.

- Σ_{arma} = {K, P, A, H, M, S, C, U, Mi}
- $\Sigma_{seleccion}$ = {g, d}
- $\Sigma_{movimiento}$ = {ADELANTE, DERECHA, ATRAS, IZQUIERDA}

- Σ_{accion} = {GOLPEAR, RECIBIR GOLPE, ESQUIVAR GOLPE}
- Σ_{estado_juego} = {go, gc, gp, eo, Ammo, -Ammo, gz, gd, -gd, gg, -gg}

El autor utilizó códigos para referir a ciertos caracteres de los alfabetos definidos. Más a detalle:

- Σ_{arma} Define el tipo de arma que posee el jugaro o enemigo. Detallando esto: K (knife), P (pistol), A (AK-47), H(hand garnade), M (machine gun), S (Sniper), C (cannon), U(Uzi) y Mi (missiles)
- $\Sigma_{seleccion}$ Define el tipo de juego que desea jugar en el nivel, g para tipo *good* y d para tipo *danger*.
- $\Sigma_{movimiento}$ El nombre de cada carácter es bastante descriptivo.
- Σ_{accion} El nombre de cada carácter es bastante descriptivo.
- Σ_{estado_juego} Cada caracter define condiciones del estado de juego. Especificando:
    - go, hace referencia a game over. Esto sucede si el poder del jugador es menor a 0.
    - gc, si los puntos del jugador es mayor a un numero en especifico. Esto se utiliza para verificar si es tiempo de enfrentar a los jefes finales.
    - gp, si los puntos del jugador es menor a un numero en especifico.
    - eo, si el enemigo murió.
    - Ammo, si el jugador tiene un tipo especifico de arma.
    - -Ammo, si el jugador no tiene un tipo especifico de arma.
    - gz, si estas en una zona en especifico.
    - gd, si nos encontramos en una zona de tipo *danger* y aun no ha sido completado.

- -gd, si nos encontramos en una zona de tipo *danger* y ya ha sido completado.
- gg, si nos encontramos en una zona de tipo *good* y aun no ha sido completado.
- -gg, si nos encontramos en una zona de tipo *good* y ya ha sido completado.

Diseño usando teoría de autómatas

En la figura 4.1 podemos ver el grafo que representa al autómata que define todas las reglas del nivel. Algunos estados utilizados son:

- **Pt**: Se le brindó al jugador algunos puntos.
- **Pow -**: Se redujo los puntos del jugador en una cantidad en especifico.
- **Pow**: Se mantiene la misma cantidad de puntos
- **Pe**: El poder del enemigo es reducido
- **He**: El escenario donde se encuentra el nivel
- **Exit Game**: El juego termina.

Análisis

Notamos que se ha usado un AFND, el cual funciona bien con el juego planteado. Puesto que al ser un juego arcade con cierta linealidad, basta con definir las acciones a suceder dependiendo de estado y caracter del alfabeto dado. Además cabe resaltar que esto mismo se puede extender a cualquier juego sea del mismo género, ya que solo es necesario saber cual sera el próximo estado como respuesta a cierta acción.

También cabe añadir la sugerencia que menciona Abbas [2], ya que si bien se usó un AFND para el diseño. En la fase de implementación es mejor usar un AFD puesto que se puede extender a código a partir de este.

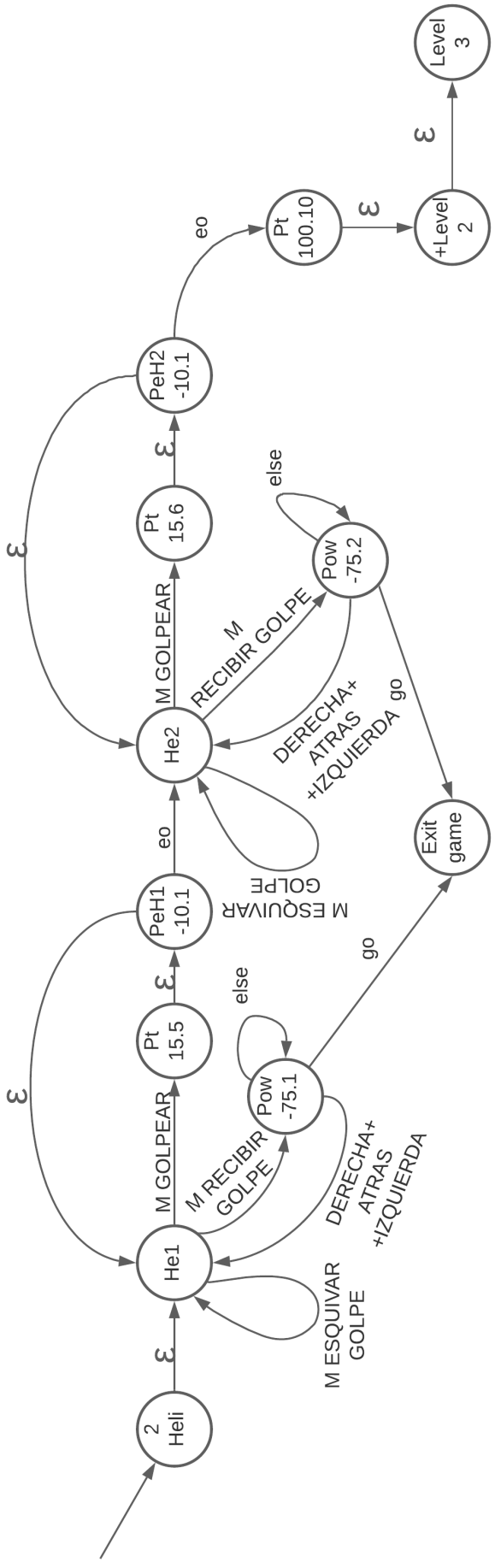

FIGURA 4.1: Nivel 2 de videojuego arcade planteado por Mushtaq [1]

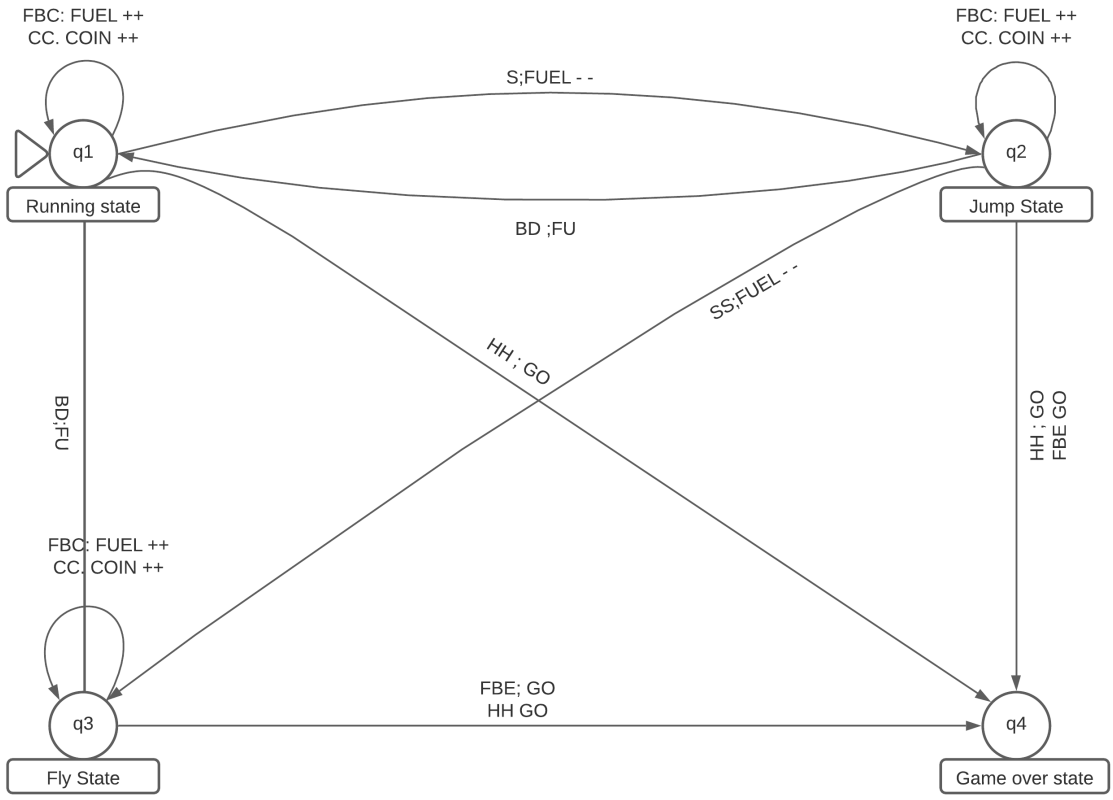

FIGURA 4.2: Diseño de un juego infinito de estilo Flappy Bird propuesto por Jamil[3] haciendo uso de Máquinas de Mealy

4.2.2. Juego infinito de tipo runner

Selección de una instancia de juego

En este caso en particular por ser un juego infinito, no es posible coger una instancia puesto que el juego termina inmediatamente solo cuando perdemos. El objetivo es alcanzar la mayor cantidad de puntos posible durando lo máximo sin perder.

A continuación se lista los alfabetos de entrada que se usaran

- $\Sigma_{movimiento}$ = {UP_ARROW, DOWN_ARROW, RIGHT_ARROW, LEFT_ARROW}
- Σ_{accion} = {S, SS}
- Σ_{estado_juego} = {GS, GE, CC, HH, FE,-F, C+, D+}

Así mismo detallamos los código utilizados por el autor del autómata.

- $\Sigma_{movimiento}$, son las entradas por teclado que se pueden dar mientras se ejecuta el juego.
- Σ_{accion}, define la acción del jugador. S si el jugador salta y SS si el jugador vuela.
- Σ_{estado_juego}, cada caracter define condiciones del estado de juego. Especificando:
  - GS, hace referencia a game starts. Sucede cuando el juego inicia.
  - GE, hace referencia a game ends. Sucede cuando el juego termina.
  - CC, hace referencia a coin collected. Sucede cuando el se recolecta una moneda.
  - FE, hace referencia a fuel ends. Sucede cuando el combustible del jugador se acaba.
  - -F, sucede cuando cierta cantidad de combustible del jugador es extraído.
  - C+, sucede cuando una moneda es añadida.
  - D+, sucede si cierta distancia es añadida.
  - HH, sucede si se golpea un obstáculo.

Diseño usando teoría de autómatas

En la figura 4.2 se puede visualizar el diseño del juego habiendo utilizado una máquina de Mealy. Los estados utlizados son:

- **Running State**: Es el estado inicial que inicializa todos los objetos en la escena de juego.
- **Jump State**: Es el estado donde saltamos y decrece la barra de salud del jugador.

- **Fly State**: Es el estado donde volamos y se reduce el combustible.
- **Game Over State**: Es el estado donde el juego termina, ocurre cuando el jugador golpea un obstáculo o enemigo.

Análisis

Observamos que el uso de la máquina de Mealy facilita la lectura del diseño en cuestión. Además que su facil lectura ayuda mucho en la comunicación programador - diseñador de juegos. La máquina de Mealy resulta ser una buena opción para juegos de este tipo, puesto que al ser infinitos tienen menos reglas definidas. Además que por ser infinito, el juego tiene objetos dinámicos como los obstáculos que aparecen aleatoriamente, en este caso una máquina de Mealy es una mejor opción puesto que esta si puede idear con objetos dinámicos a diferencia que un AFD o AFND.

4.2.3. Juego de Pacman

Selección de una instancia de juego

Por ser el rediseño de un juego bastante conocido, se utilizara un nivel estándar del videojuego. Puesto que este solo cuenta de paredes, puntos, el jugador, potenciador y enemigos fantasmas. El juego consiste en comer todos los puntos sin morir al chocar con los enemigos. El potenciador sirve para inhabilitar a los enemigos por un tiempo limitado, solo en este tiempo el jugador puede chocar a los enemigos.

Los caracteres que permiten las transiciones son bastante descriptivos, por lo que no se detallará como se hizo en los apartados anteriores.

Diseño usando teoría de autómatas

En la figura 4.3 se puede visualizar el diseño del juego habiendo utilizado un autómata de pila. Los estados utilizados pueden no ser muy descriptivos, por lo que se detallara a continuación:

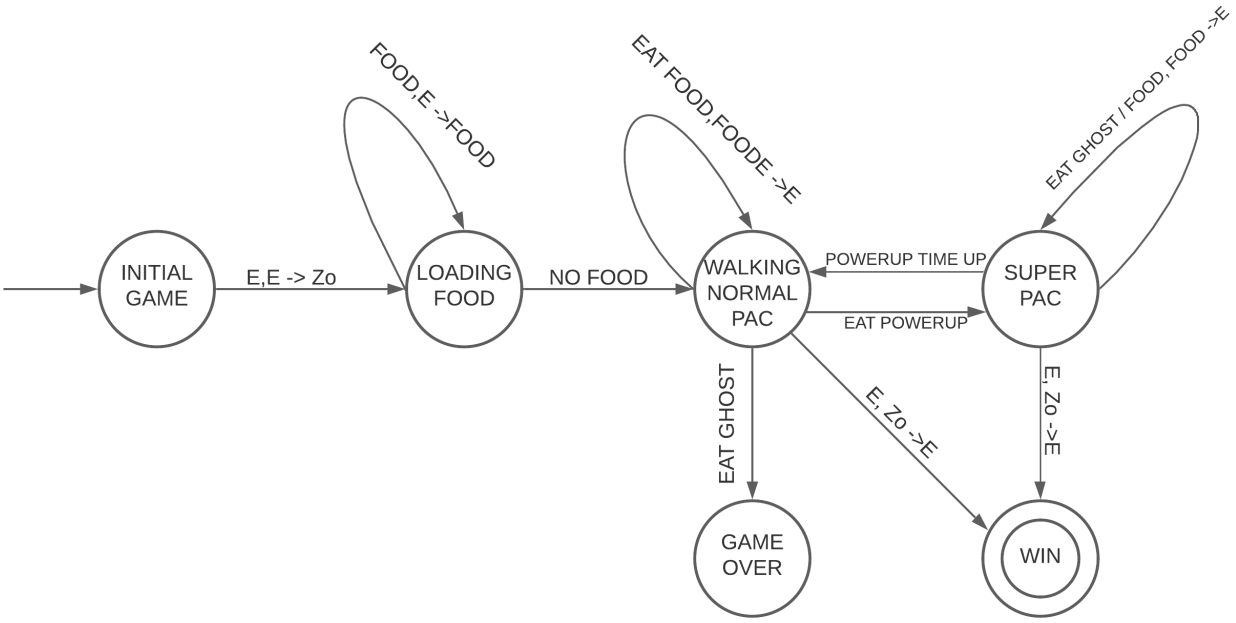

FIGURA 4.3: Rediseño del juego Pacman haciendo uso de un autómata de Pila

- **Initial Game**: Es el estado inicial. En este estado se cargan todos los elementos del juego.
- **Loading Food**: En este estado se ubican los puntos dentro del escenario. Esto varia de nivel a nivel puesto que todos los escenarios no son los mismos.
- **Walking Normal Pac**: Es el estado en el que transcurre la mayor parte del tiempo para el jugador. Consiste en caminar y comer los puntos.
- **Super Pac**: Es el estado en el que se encuentre el jugador después de haber comido el potenciador.
- **Game Over**: Es el estado de fin de juego, ocurre cuando no tenemos un potenciador y chocamos con un enemigo.
- **Win**: Es el estado final del juego, ocurre cuando todos los puntos han sido comidos.

Análisis

El uso del autómata con pila esta justificado por la necesidad de memorización.Esto es utilizado al memorizar todos los puntos que han sido comidos, así como también facilita la tarea de determinar si el juego ha sido ganado.

Esto nos muestra la importancia de tener variedad al momento de determinar el uso de un tipo de autómata. Además se puede afirmar que en el diseño de juegos que necesiten memorización, el uso de un autómata con pila es útil. Un ejemplo de estos puede ser el genero de juegos de mesa, puesto que estos necesitan memorización en su desarrollo.

Capítulo 5

Conclusiones y trabajos futuros

5.1. Conclusiones

Después de haber desarrollado el contenido de la presente tesis se llegaron a las siguientes conclusiones:

- Es posible realizar el diseño de un juego utilizando la teoría formal de autómatas.
- El diseño basado en autómatas puede ayudar el proceso de implementación ya que este estará mas sistemático y organizado.
- El diseño de juegos basado en autómatas es una alternativa a las herramientas clásicas de la ingeniería de software, como los diagramas de caso de uso.
- Si se usa el diseño basado en autómatas, se debe evitar el uso de AFND puesto que esto complica la implementación directa.

5.2. Trabajos futuros

El diseño de juegos basado en autómatas abre muchas posibilidades ya que su estudio no ha sido demasiado explorado. A continuación se listaran algunos posibles trabajos a futuro que surgen a apartir de este:

- El diseño de juegos basado en autómatas no tiene por que limitarse solamente a videojuegos. Sino también, puede extenderse al diseño de

software, ya que como vimos el uso de autómatas puede reemplazar algunas herramientas de la ingeniería de software.

- La implementación y documentación del proceso de desarrollo de un videojuego desde cero, haciendo uso de la metodología presentada en la presente tesis.
- La exploración de otros tipos de autómatas haciendo uso de otras herramientas como inteligencia artificial para el diseño de juegos.
- El estudio y el diseño de mayor cantidad de juegos de distintos géneros para poder hacer una clasificación más precisa de autómata-género de juegos.

Bibliografía

[1] Hassan Mushtaq. Computing game design with automata theory. *International Journal of Multidisciplinary Sciences and Engineering*, 2012.

[2] Noman Sohaib Qureshi & Zahid Abbas & Muhammad Sohaib & Muhammad Arshad & Rizwan Ali Sabir & Asma Maqsood. A roller coaster game design using automata theory. *International Journal of Multidisciplinary Sciences and Engineering*, 2012.

[3] Abid Jamil & Asad Ullah & Mohsin Rehman. An infinite runner game design using automata theory. *International Journal of Computer Science and Software Engineering*, 2016.

[4] Ariana Yunita & Riestiya Zain Fadillah & Muhammad Redho Darmawan. Re-designing the pacman game using push down automata. *Asia Pacific Conference on Contemporary Research*, 2016.

[5] John E. Hopcroft and Jeff D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[6] Jesse Schell. *The Art of Game Design: A book of lenses*. Morgan Kaufmann, 2008.