Lenguajes, Computación y Sistemas Inteligentes

Informe de proyecto

Implementación de Máquinas Turing en

Baldwin David Rodríguez Ponce

9 de enero de 2020

Índice

Introducción

Desde hace ya unos años, los videojuegos han sido una parte importante del entretenimiento de las personas. La gran variedad de géneros disponibles permiten que cualquiera pueda encontrar algo que le resulte entretenido e interesante. Sin embargo, cabe destacar que se han dado casos en los que estos videojuegos han mostrado la habilidad de hacer más que sólo entretener. Cuando un jugador tiene la posibilidad de crear sus propios niveles dentro de un juego, uno se puede encontrar con todo tipo de sorpresas.

Durante este proyecto hablaremos de un videojuego en específico: "Baba Is You". Éste mismo pertenece a un pequeño grupo de juegos que, no solamente le permiten al jugador mostrar su creatividad, sino que son tan potentes como un lenguaje de programación. Para explicar esta potencia, revisaremos conceptos de teoría de la computación y explicaremos lo que es "Baba Is You" como videojuego y sus principales características.

Una vez explicados estos conceptos, veremos cómo se pueden implementar máquinas y autómatas de Turing dentro del videojuego. Esto con el objetivo de demostrar que "Baba Is You" tiene todo lo necesario para ser considerado un lenguaje de programación por sí mismo.

1. Marco Teórico

Antes de comenzar a hablar sobre cómo se implementa una máquina de Turing dentro de "Baba Is You", conviene conocer los conceptos y definiciones pertenecientes al tema de teoría de computabilidad y al juego en sí. Empezaremos, por tanto, explicando cada uno de ellos con el objetivo de hacer más fácil el seguimiento del proyecto y tener una base teórica sobre la cual trabajar.

1.1. Computabilidad

La teoría de la computabilidad es la parte de la computación que estudia los problemas que se pueden resolver con un algoritmo. Sin embargo, nosotros no profundizaremos demasiado en el tema por dos razones: primero, para este proyecto basta con conocer tres conceptos de este campo y; segundo, se presupone que el lector ya tiene cierta familiaridad con este mismo. Habiendo dicho esto, continuamos con la explicación.

1.1.1. Máquinas y Automátas de Turing

Una máquina de Turing es un dispositivo que manipula símbolos sobre una cinta infinita dependiendo de una serie de estados internos que lo condicionan. Decimos que dicha máquina devuelve un resultado que empieza desde una posición de la cinta y termina en otra. Podemos considerar a los autómatas de Turing como una modificación de una máquina de Turing en la que se especifican los estados finales de la máquina y sólo existen dos resultados posibles: aceptado y rechazado. Podemos definir ambos de la siguiente forma:

- Automáta: Es una 6-tupla M tal que $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ con $\delta : Q \times (\Sigma \cup \Gamma) - \rightarrow Q \times (\Sigma \cup \Gamma) \times \{R, L\}$ y se aceptan las palabras por estado final.
- Máquina: Es una 5-tupla M tal que $M = (Q, \Sigma, \Gamma, \delta, q_0)$ con $\delta : Q \times (\Sigma \cup \Gamma) - \rightarrow Q \times (\Sigma \cup \Gamma) \times \{R, L\}$ y se aceptan las palabras por parada de la máquina.

Siendo:

- Q el conjunto de estados de la máquina/autómata de Turing.
- Σ es el alfabeto de entrada. En él se encuentran los símbolos propios de la entrada y la salida del cómputo.
- Γ es el alfabeto de cinta. En él se encuentran los símbolos propios de la cinta y no de la salida o entrada del cómputo.
- δ es la función de transición.
- q_0 es el estado inicial. Se cumple que $q_0 \in Q$
- F es el conjunto de estados finales del autómata. Se cumple que $F \subseteq Q$

Es necesario mencionar que, en nuestra definición, $\square$ es el único símbolo que se repite de forma infinita en toda la cinta ($\square \in \Gamma$); es decir, el símbolo vacío o en blanco.

1.1.2. Tesis de Church-Turing

La tesis de Church-Turing expone que todo algoritmo es equivalente a una máquina de Turing (o a cualquier máquina de cálculo). No se considera un teorema matemático, sino que una afirmación formalmente indemostrable; sin embargo, tiene una aceptación casi universal. Decimos que es indemostrable ya que nunca se ha podido diseñar un computador que implemente algoritmos intraducibles a una máquina de Turing y nunca se ha encontrado un modelo teórico de computación que sea más potente que éstas mismas.

De esta tesis salen las siguientes implicaciones:

- Todo sistema de computación es a lo sumo tan potente como las máquinas de Turing.
- Si una función no es Turing-computable, tampoco se puede ni se podrá encontrar una solución o algoritmo para la misma en ningún sistema de computación.

1.1.3. Función Computable

Una función computable es una formalización de la noción de algoritmo. Según la Tesis de Church-Turing, las funciones computables son exactamente las funciones que pueden ser calculadas con una máquina de cálculo (una Máquina Turing, en nuestro caso).

La función calculada por una máquina de Turing M se denota φ_M

$$\text{Siendo } \varphi_M(X_1, X_2, ..., X_n) \cong \begin{cases} y, & \text{si } (\varepsilon, q_0, X_1\#X_2\#...\#X_n) \vdash^* (u, q, ysv) \wedge s \notin \Sigma \\ \perp, & \text{c.c.} \end{cases}$$

Una función Ψ se dice que es Turing-computable si existe una máquina de Turing M que la calcula; es decir, existe φ_M tal que $\varphi_M \cong \Psi$. Por ende, cualquier función (problema) entre cadenas de símbolos que admita un algoritmo de cualquier tipo es Turing-computable. Se puede decir lo mismo de cualquier función entre objetos de cualquier tipo de datos que se puedan implementar mediante cadenas de símbolos. También, gracias a esta definición, podemos decir que las funciones computables son infinitamente enumerables; existiendo, por tanto, más funciones no computables que computables.

La noción de Turing-computabilidad no es específica de las máquinas de Turing. Todos los lenguajes, ordenadores y entornos de programación conocidos computan exactamente las mismas funciones.

1.2. Baba Is You

Habiendo expuesto los conceptos necesarios de teoría de computabilidad, pasamos a explicar qué es exactamente el videojuego que usaremos en el proyecto. "Baba Is You" es un juego de rompecabezas y lógica creado por el desarrollador finlandés Arvi Teikari (conocido también como Hempuli). El juego fue creado durante el concurso de creación de videojuegos Nordic Game Jam en el 2017, ganando el primer lugar en dicho concurso y consiguiendo buenas críticas por parte de los participantes. Esto último sirvió como impulso para que el creador decidiera hacer un lanzamiento oficial del juego, siendo éste publicado el 13 de marzo de 2019 en tiendas virtuales.

El juego se centra en la manipulación de los objetos de cada nivel a través de la interacción de bloques que representan las reglas de este mismo. El objetivo es que el jugador, quien usualmente controla al personaje llamado Baba, supere el nivel interactuando con estos bloques.

1.2.1. Jugabilidad

La principal característica del juego, como hemos mencionado antes, es la manipulación del entorno a través de lo que "Baba Is You" llama "palabras". Para resolver cada rompecabezas, se deben manipular las palabras con las que se inicia el nivel con el fin de poder crear nuevas reglas y llegar a la solución. Por ejemplo, las tres palabras más conocidas son "*Baba*", "*is*" y "*You*"; al ponerlas juntas de manera que se pueda leer la oración "*Baba is You*" el jugador gana la posibilidad de controlar el personaje conocido como Baba. Si se llegase a mover una de las palabras de manera que la oración se rompa, el jugador perdería la posibilidad de controlar a Baba y esto en muchas situaciones supone perder la partida. Llamaremos "reglas" a dichas oraciones.

El juego, sin embargo, hace distinción entre el tipo de palabras que se están usando, clasificándolas en las siguientes tres categorías:

1. Sustantivos: Palabras que representan objetos del juego. Llamamos objetos a todo aquello que tenga un sprite o imagen asignada dentro del nivel. Por ejemplo: "*Baba*", "*Tree*" y "*Water*".

2. Propiedades: Palabras que al ir acompañados de un sustantivo y un operador, modifican el comportamiento del objeto representado por el sustantivo dependiendo de lo que indique el operador. Por ejemplo: "*You*"(hace que el jugador controle al objeto), "*Blue*"(hace que el objeto se vuelva azul) y "*Weak*"(hace que el objeto desaparezca al entrar en contacto con otros objetos).

3. Operadores: Palabras que representan relaciones entre los sustantivos y las propiedades. Dentro de este tipo de palabras se encuentran otros subtipos que deben ser mencionados:

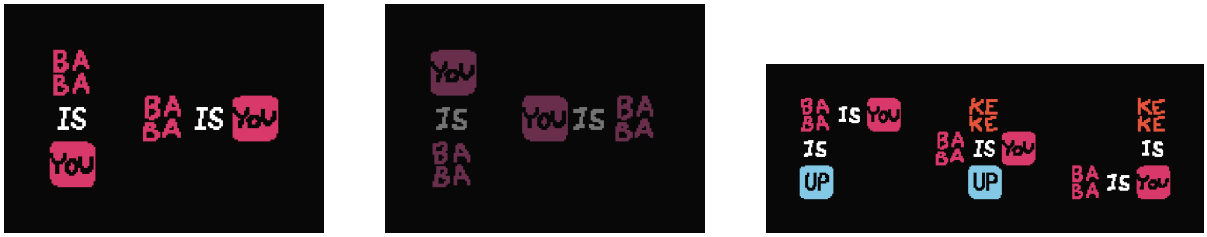
   a) Verbos: Palabras que se usan para indicar el tipo de relación que tienen los objetos y las propiedades adjacentes a ellos. Por ejemplo: "*Baba is Blue*" hará que Baba se vuelva azul y "*Baba make Rock*" hará que aparezca una roca en la casilla donde se encuentra Baba cada vez que se mueva.

   b) Condicionales: Se pueden poner al lado de una sola palabra (adjetivos y el negativo) o se pueden usar para decir que un objeto debe cumplir una cierta condición en función de otro objeto (preposiciones) para que el verbo anexado funcione. Por ejemplo: "*Baba on Rock is Blue*" hará que Baba se vuelva azul solamente si se encuentra en la misma casilla donde está una roca.

   c) Conjunción: Conocido como "*and*" dentro del juego, este operador junta dos palabras de forma que puedan ser usadas al mismo tiempo con un sólo verbo. Por ejemplo: "*Baba and Rock is Blue*" equivale a formar las reglas "*Baba is Blue*" y "*Rock is Blue*" por separado.

1.2.2. Reglas

"Baba Is You" nos permite formar reglas para manipular el entorno. Sin embargo, no cualquier combinación de palabras forma una regla válida. Debemos conocer las restricciones que impone el juego para la formación de dichas reglas. Una de estas restricciones para formar reglas válidas es que la combinación de palabras, al ser leída de izquierda a derecha o de arriba a abajo, se puedan leer como una oración. Por ejemplo, en la figura 1.a aparecen reglas válidas; mientras que en la figura 1.b, no. Existe también la posibilidad de cruzar reglas y hacer que compartan una palabra gracias a esto (figura 1.c) y este cruce no afecta la validez de una regla.



(a) Reglas válidas (b) Reglas inválidas (c) Reglas que comparten palabras

Figura 1: Ejemplos de combinaciones de palabras

Otra de las restricciones está ligada al uso de operadores condicionales para restringir una regla y la conjunción para fusionar reglas. Ejemplo de esto lo podemos observar en la figura 2. En este ejemplo tenemos primero en la figura 2.a cuatro reglas que afectan a los objetos "*Baba*" y "*Door*"; usando la conjunción podemos combinar ambos como se muestra en la figura 2.b y obtener dos reglas que funcionan igual. Cuando nosotros decimos "*Door is Open and Stop*", estamos haciendo que el comportamiento de la puerta dependa de esas propiedades en cada momento; sin embargo, nosotros podemos hacer también que dependa sólo en determinados momentos. Cuando usamos el operador condicional "*on*" podemos hacer que las propiedades de la regla sólo cambien el comportamiento de la puerta cuando Baba se encuentre en la misma casilla.

(a) Reglas separadas

(b) Reglas fusionadas con la conjunción

(c) Regla con condicional

Figura 2: Operadores cambiando la forma y el comportamiento de una regla

Aunque el ejemplo de la figura 2 pudiera hacer creer que la conjunción sólo funciona con propiedades, ése no es el caso. La conjunción se puede usar para fusionar dos reglas que comparten un sustantivo o una propiedad. Aunque esto siempre ligado al hecho de que la fusión no cambiará lo que ambas reglas imponen por separado (el mero hecho de que dos reglas compartan una palabra no es suficiente para usar una conjunción y esperar un comportamiento equivalente).

1.2.3. Lista de palabras

Dentro de este videojuego podemos encontrar una larga lista de palabras; sin embargo, nosotros solamente explicaremos algunas cuantas necesarias para el proyecto. Como hemos mencionado antes, los sustantivos son palabras que representan objetos en el juego, por esa misma razón los excluiremos de esta lista y nos centramos sólo en algunos operadores y propiedades. Describiremos la funcionalidad de estas mismas usando palabras genéricas que denotaremos como $sust_i$ (para los sustantivos), $verb_j$ (para los verbos), $prop_k$ (para las propiedades).

Sin embargo, vamos a hablar primero de un sustantivo un tanto especial: "*Empty*". Podemos considerar a este sustantivo como una pequeña excepción a lo que dijimos antes ya que el objeto que representa es más abstracto. "*Empty*" representa el vacío dentro del juego. Cuando una casilla no tiene nada encima de ella, la consideramos "*Empty*". Si tuviésemos un nivel con nada dentro de él excepto la regla "*Empty is Baba*",

en todos los espacios vacíos aparecería inmediatamente una instacia de Baba. Así mismo, si tenemos la regla "*Baba is Empty*", todas las instancias de Baba desaparecen del nivel. Habiendo explicado esto, podemos proceder con los operadores y propiedades.

En los operadores tenemos:

1. *is*: Verbo. Este operador es el más usado en todo el juego y podemos usarlo con dos objetivos distintos:
   a) Cuando se junta un sustantivo con el verbo "*is*" seguido de una propiedad, el comportamiento de dicho objeto cambia en base a la propiedad, un ejemplo de esto ya ha sido mencionado antes ("*Baba is Blue*").
   b) Cuando se junta un sustantivo con el verbo "*is*" seguido de otro sustantivo, el objeto del primer sustantivo se transforma en el objeto del segundo.
2. *make*: Verbo. Este operador sólo funciona cuando se junta con dos sustantivos. Cuando formamos la regla "$sust_1$ *make* $sust_2$", todos los objetos representados por el primer sustantivo crearán una instancia del objeto representado por el segundo sustantivo en la casilla donde se encuentran.
3. *on*: Condición de tipo preposición. Esta palabra requiere que se junten dos sustantivos. Cuando formamos la regla "$sust_1$ *on* $sust_2$ $verb_1$ $prop_1$" o la regla "$sust_1$ *on* $sust_2$ $verb_1$ $sust_3$", estamos diciendo que la acción del verbo sólo ocurrirá si una instancia del primer sustantivo está en la misma casilla que una instacia del segundo.
4. *and*: Conjunción. Este operador funciona para fusionar reglas, como hemos mencionado antes. Por ejemplo, si en el nivel están las reglas "$sust_1$ $verb_1$ $prop_1$" y "$sust_2$ $verb_1$ $prop_1$" podemos crear la regla equivalente "$sust_1$ *and* $sust_2$ $verb_1$ $prop_1$" (nótese que el verbo y la propiedad en ambas reglas son iguales).

Y en las propiedades:

1. You: Hace que el objeto de la regla pueda ser controlado por el jugador. Para que un nivel sea completado correctamente debe existir como mínimo un objeto con esta propiedad.
2. $Right$, $Left$, Up: Modifican la dirección del objeto.
3. $Move$: Hace que el objeto de la regla se mueva en la dirección que tiene asignada. Por defecto, los objetos están viendo hacia abajo.
4. $Hide$: Hace que el objeto de la regla no se pueda ver en el nivel, pero siga funcionando como si no estuviera oculto.
5. Win: Hace que el objeto de la regla se convierta en el objetivo del nivel. Cuando un objeto con esta propiedad toca al personaje del jugador, el nivel se considera completado.
6. $Defeat$: Cuando un objeto con esta propiedad toca al personaje del jugador, el personaje desaparece. En muchas situaciones, esto supone perder el nivel.

2. Desarrollo

2.1. Implementación

Nuestro objetivo, como hemos mencionado antes, es implementar una máquina de Turing dentro de "Baba is You". Sin embargo, podemos reducir el problema a algo más sencillo: ¿Cómo podemos implementar dos estados, los símbolos que utiliza la cinta y la transición que ocurre entre ellos de forma general? Si logramos solucionar este problema, podremos aplicar su solución a cualquier tipo de transición que ocurre entre dos estados cualesquiera y, por extensión, implementar cualquier máquina de Turing.

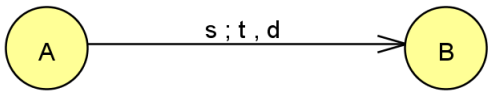


Figura 3: Transición entre dos estados de una máquina de Turing

Tomando todo lo mencionado anteriormente, podemos centrarnos solamente en buscar una forma de implementar la máquina de Turing que se muestra en la figura 3 sabiendo que $s, t \in (\Sigma \cup \Gamma)$ y $d \in \{R, L\}$.

Como hemos mencionado antes, "Baba is You" permite manipular los objetos de su entorno a través de la manipulación de las palabras que los representan. Por ende, utilizaremos objetos propios del juego para simular los elementos de la máquina de Turing.

2.1.1. Implementación de los estados y los símbolos

Para cada estado O de forma que $O \in Q$ requeriremos tres objetos (junto con las palabras que los representan) en el juego de forma que los tres existan casi simultáneamente (explicaremos más adelante por qué decimos "casi"). Estos objetos los nombraremos en base a su función:

- Objeto de Lectura (O_L): Se encargará de leer el símbolo al que apunta la cabeza lectora.
- Objeto de Desplazamiento/Movimiento (O_M): Se encargará de moverse en la dirección especificada en la transición.
- Objeto de Escritura (O_E): Se encargará de cambiar el símbolo al que apunta la cabeza lectora.

Usaremos la colección de objetos resultantes de todos los estamos como una forma de representar la cabeza lectora. Impondremos la restricción de que sólo puedan existir los objetos de un estado específico durante la lectura de un símbolo y diremos que el objeto de lectura que se observa en el juego representa el estado actual de la cabeza lectora y su posición en la cinta.

Así mismo, utilizaremos un objeto por cada símbolo de $\Sigma \cup \Gamma$. Ubicaremos estos objetos en una fila para representar la cinta y los cambios de estos mismos ocurrirán al interactuar con los objetos que representan los estados de la máquina de Turing.

2.1.2. Implementación de la lectura y la escrita

La implementación de la lectura y la escritura la podemos simular utilizando una serie de reglas en base a los objetos que hemos definido antes. En la figura 3 tenemos dos estados, entonces para cada uno de ellos tendremos su respectivo objeto de lectura, movimiento y escritura ($A_L,\ A_M,\ A_E,\ B_L,\ B_M,\ B_E$). Sabiendo esto podemos definir las reglas dentro del juego:

- A_L *on* s *is* d *and* B_M: Está regla nos dice que si el objeto de lectura de A se encuentra sobre s, entonces este objeto se transforma en el objeto de movimiento de B y le asigna la dirección d hacia donde se moverá. Con esta regla simulamos el cambio de estado de la cabeza lectora.
- A_L *make* A_E: El objeto de lectura de A pone su objeto de escritura en la misma casilla donde se encuentra.
- s *on* A_E *is* t: El símbolo s, al ponerse en contacto con el objeto de escritura de A, cambia inmediatamente al símbolo t. De esta forma simulamos la escritura en la cinta de la máquina de Turing.

Utilizando estas tres reglas podemos simular los cambios de estado y del símbolo donde se encuentra la cabeza lectora.

2.1.3. Implementación del movimiento

Hasta ahora sólo hemos hecho uso de los objetos de escritura y lectura de un estado en específico; pero como vimos en el apartado anterior, existe una regla que hace que el objeto de lectura de A se convierta en el objeto de movimiento de B. La razón por la cual hacemos esto es que, para hacer funcionar correctamente la máquina de Turing, hacemos que todos los objetos de movimiento estén siempre moviéndose. De esta forma sólo debemos encargarnos de indicar en qué dirección deben moverse y de decidir cuándo pueden detenerse a leer el siguiente símbolo. Tomando esto en cuenta, para implementar correctamente el movimiento de la cabeza lectora, utilizamos las siguientes reglas:

- B_M *is* $Move$: Esta regla es la que hace que el objeto de movimiento siempre se mueva en la dirección que se le ha específicado.
- B_M *is* B_L: Una vez se haya movido el objeto, esta regla se encargará de transformar el objeto de movimiento de B en su objeto de lectura.

2.1.4. Finalización del cómputo

Para implementar la finalización del cómputo de nuestra máquina, debemos buscar una forma de implementar el caso en el que el cómputo termina sin problemas y el caso en el que falla por leer un símbolo que no debería haber leído. Hasta ahora nos hemos estado centrando solamente en la implementación de máquinas y no de autómatas de Turing. La razón de esto es que la implementación de ambos es exactamente la misma excepto en su finalización.

Decimos que la máquina de Turing ha terminado su cómputo cuando llega a un estado del cual no puede moverse a ningún otro (en el caso de los autómatas, cuando llega a un estado final y no tiene más símbolos que leer). Para poder implementar el caso en el que la máquina termina, utilizaremos un objeto extra propio del juego: la bandera. Una vez llegado a un estado que podamos considerar como el último del cómputo, haremos que este se convierta en la bandera (en el juego "Flag"). Usaremos la siguiente regla junto con esto:

$$Level\ on\ Flag\ is\ Win$$

Esta regla hará que el nivel sea completado en el momento que aparezca la bandera en el nivel y podremos decir que el cómputo ha sido realizado con éxito. En el caso de que estemos usando una máquina de Turing, el resultado será lo que queda en la cinta desde la cabeza lectora hasta la primera aparición de un símbolo de Γ; mientras que en el caso de que estemos usando un autómata, simplemente diremos que la palabra ha sido aceptada si el nivel termina.

Como es normal, si tenemos una forma de decir que la palabra ha sido aceptada, también tenemos que buscar una forma de decir que la palabra ha sido rechazada. Cuando un autómata o una máquina de

Turing lee un símbolo que no se esperaba leer, haremos que el estado en el que se encuentra cuando lee el símbolo se convierta en un objeto que llamaremos "Stop". Usaremos la siguiente regla junto con este objeto:

$$Level\ on\ \ is\ Defeat$$

Esta regla hará que el jugador falle el nivel una vez aparezca el objeto "Stop". En el caso de estar usando una máquina de Turing, diremos que su resultado ha sido indefinido; mientras que en el caso del autómata, diremos que la palabra ha sido rechazada.

2.1.5. Funcionalidades y detalles extras

A pesar de que nos bastan las reglas mencionadas en los apartados anteriores para poder implementar cualquier transición entre dos estados de una máquina de Turing, nosotros implementaremos unas reglas extras para facilitar la visualización y la finalización de la transición. Estas reglas se aplicarán siempre a todos los estados de la máquina y no dependerán del símbolo en el que se encuentran.

Como hemos mencionado antes, los objetos para representar cada estado se mueven juntos. Debido a que cada objeto tiene su propia imagen dentro del juego, tenerlos uno encima de otro puede complicar su visualización. Por esta razón, añadimos la siguiente regla:

$$A_L\ and\ B_L\ is\ Hide$$

Esta regla nos permitirá esconder los objetos de lectura sin eliminarlos del juego. Lo único que ocurrirá es que no podremos verlos en la cinta pero seguirán funcionando como antes. Sin embargo, estos objetos generan sus propios objetos de escritura. Dichos objetos sólo son útiles durante un turno para hacer que el símbolo donde se encuentran cambie, pero si no nos deshacemos de ellos inmediatamente después de que hayan realizado este cambio, se quedarán en el mismo lugar y causarán un comportamiento inesperado dentro de la máquina. Por esta razón implementamos la siguiente regla:

$$A_E\ and\ B_E\ is\ Empty$$

Esta regla se encargará de convertirlos en objetos nulos ("Empty"), dejando sólo el símbolo con el cual compartían espacio.

Por último, nos gustaría mencionar ciertos cambios y limitaciones causadas por el juego en sí. A pesar de que "Baba is You" permite cambiar las imágenes y colores de los objetos, no hay mucha variedad de la cual elegir para representar cada uno. Por esta razón se ha decidido hacer que todos los objetos que representen estados luzcan como burbujas transparentes y a cada estado se le ha asignado un color distinto como se puede observar en la figura 4. Si consideramos la burbuja como una sola entidad, podemos decir que ésta es la cabeza lectora. Cuando la burbuja cambia de color, decimos que ha cambiado de estado.

Figura 4: Palabras de los objetos que representan los estados de una máquina de Turing

También es necesario mencionar que para facilitar la distinción entre estados, al grupo de tres palabras que representan un estado se les ha puesto como imagen una letra única. Para distinguir entre los tipos de objetos se ha hecho que los objetos de lectura tengan un color intenso, los objetos de escritura un color oscuro muy desaturado y los objetos de movimiento un color ni muy intenso ni muy desaturado.

2.1.6. Generalización

Ahora que ya sabemos cómo simular cada parte de la máquina de Turing dentro de Baba is You, podemos pasar a dar un modelo general para implementar cualquier máquina que deseemos. Para cualquier Máquina de Turing M con la definición de Máquina de Turing presentada en el marco teórico, usaremos los siguientes pasos con el fin de implementarla:

- Reservamos un objeto para cada $s \in (\Sigma \cup \Gamma)$
- Reservamos tres objetos para todo $O^{i)} \in Q$, estos objetos los denotaremos como $O_L^{i)}$, $O_M^{i)}$ y $O_E^{i)}$.
- Añadiremos las siguientes reglas para los n estados dentro de Q:
  1. O_L *and* O'_L *and* O''_L *and ...* $O_L^{n)}$ *is* $Hide$. Esta regla se encargará de ocultar el objeto de lectura.
  2. O_M *and* O'_M *and* O''_M *and ...* $O_M^{n)}$ *is* $Move$. Esta regla se encargará de mover los objetos de movimiento.
  3. O_E *and* O'_E *and* O''_E *and ...* $O_E^{n)}$ *is Empty*. Esta regla se encargará de eliminar los objetos de escritura una vez hayan sido utilizados.
- Cualquier transición $\delta(O, s) = (O', t, d)$ con $s, t \in (\Sigma \cup \Gamma)$, $d \in \{R, L\}$ y $O, O' \in Q$ se puede implementar con las siguientes reglas dentro del juego:
  1. O_L *on* s *is* d *and* O'_M. Esta regla simula la escritura y la asignación de una dirección.
  2. s *on* O_E *is* t. Esta regla simula la escritura.
  3. O_M *is* O'_L. Esta regla se encarga de detener la cabeza lectora una vez se ha movido una casilla.
  4. O_L *make* O_E. Este objeto se encarga de crear los objetos de lectura.
- Si en un determinado estado $O \in Q$ leemos el símbolo $s \in (\Sigma \cup \Gamma)$ y sabemos que $\nexists O' \in Q(\delta(O, s) = (O', t, d))$ con $t \in (\Sigma \cup \Gamma)$ y $d \in \{R, L\}$, entonces hemos leído un símbolo inesperado. Añadiremos la regla "O *on* s *is Up and Stop*" en este caso para hacer que el cómputo termine.
- Dados $s, t \in (\Sigma \cup \Gamma)$ y $d \in \{R, L\}$, por cada $O \in Q$ tal que $\nexists O' \in Q[\delta(O, s) = (O', t, d)]$, añadiremos la regla "$O_L$ *is Up and Flag*" para indicar que este estado termina el cómputo. En realidad podríamos obviar la asignación de una nueva dirección, pero se ha decidido añadirla para que la cabeza lectora no pueda moverse más hacia ningún otro lado.
- Añadiremos la regla "*Level on Flag is Win*" para que, al momento de terminar el cómputo, el nivel acabe.
- Por último, añadiremos la regla "*Level on Stop is Defeat*" para aquellos casos en los que la palabra leída tenga un símbolo inesperado. Este es nuestro equivalente a decir que el resultado de la función de esta máquina ha sido indefinido.

Y si lo que queremos implementar es un autómata de Turing usando la definición presentada en el marco teórico, simplemente sustituiremos los tres últimos pasos anteriores por los siguientes:

- En caso de que nos encontremos en un estado $O \in F$ de forma que leamos el símbolo $\square$, podremos decir que hemos terminado el cómputo y que la palabra ha sido aceptada. Para estos casos añadimos la regla "O *on* $\square$ *is Up and Flag*".
- Añadimos la regla "*Level on Flag is Win*" para poder terminar el nivel en aquellos casos en los que el autómata acepte la palabra.
- Por último, añadimos la regla "*Level on Stop is Defeat*" para aquellos casos en los que la palabra tenga un símbolo que haga que el autómata la rechace.

Las diferencias entre un autómata de Turing y una máquina de Turing en esta implementación son sólo de interpretación. Para la máquina de Turing, que el nivel sea completado o no sólo determina si existe una solución definida con la palabra usada para computar; en cambio, para el autómata, esto indica si la palabra ha sido rechazada o no.

Usando estos pasos, ahora somos capaces de implementar tanto máquinas como autómatas de Turing en "Baba Is You".

2.2. Ejemplos de Máquinas de Turing construidas en el juego

Utilizaremos los pasos mencionados anteriormente para implementar dos máquinas de Turing y un autómata de Turing como demostración. Para estos ejemplos usaremos siempre el mismo alfabeto de entrada $\Sigma = \{0, 1\}$ y, por lo general, el mismo alfabeto de cinta $\Gamma = \{\square\}$. Debido a las limitaciones que presenta usar un documento para ver a las máquinas funcionar, añadiremos hipervínculos en los anexos para poder ver grabaciones de éstas desde internet.

Como se ha mencionado antes, la cinta de la máquina será una sucesión de los objetos que representan a los símbolos de los alfabetos. Un ejemplo de esto se puede ver en la figura 5. Cabe destacar que, como "Baba Is You" no tiene ninguna imagen similar al símbolo $\square$, hemos decidido usar el símbolo X para representarlo.

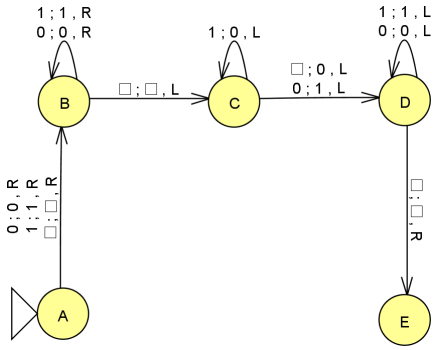
Figura 5: Ejemplo de una palabra sobre la cinta de la máquina

2.2.1. Siguiente(x)

Crearemos una máquina de Turing que compute la palabra siguiente de otra dentro del alfabeto especificado anteriormente. La función de esta máquina la podemos definir inductivamente sabiendo que $\forall w \in \Sigma^*$:

$$\varphi_M(x) \widetilde{=} siguiente(x) = \begin{cases} 0, & \text{si } x = \varepsilon \\ w \cdot 1, & \text{si } x = w \cdot 0 \\ sig(w) \cdot 0, & \text{si } x = w \cdot 1 \end{cases}$$

La máquina de Turing que usaremos será $M = (\{A, B, C, D, E\}, \Sigma, \Gamma, \delta, A)$:



(a) Esquema

δ	□	0	1
->A	(B,□,R)	(B,0,R)	(B,1,R)
B	(C,□,L)	(B,0,R)	(B,1,R)
C	(D,0,L)	(D,1,L)	(C,0,L)
D	(E,□,R)	(D,0,L)	(D,1,L)
E	-	-	-

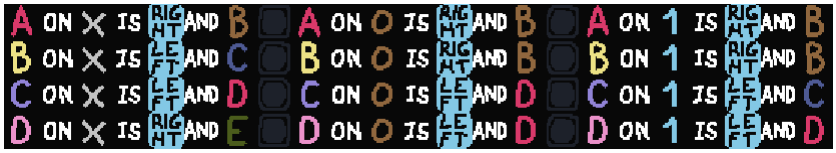
(b) Función de transición

Figura 6: Máquina de Turing que cómputa la siguiente palabra de otra

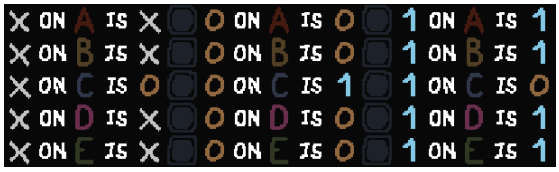
Primero reservaremos un objeto del juego para cada uno de los símbolos $s \in (\Sigma \cup \Gamma)$ y tres objetos para cada $O \in Q$. Acto seguido crearemos las reglas que afectan a los objetos de cada estado de la máquina de Turing. Estas reglas las podemos ver en la figura 7.

Figura 7: Reglas de movimiento, limpieza y ocultación

Después, utilizando la función de transición, implementaremos cada una de las reglas necesarias para simular la lectura y la escritura:



(a) Lectura y dirección



(b) Escritura

(c) De M a L y Generador de E

Figura 8: Reglas que controlan la lectura y escritura de “Siguiente(x)”.

Por último, añadiremos la sucesión de símbolos que representan las reglas que hacen que termine.

(a) Finalización

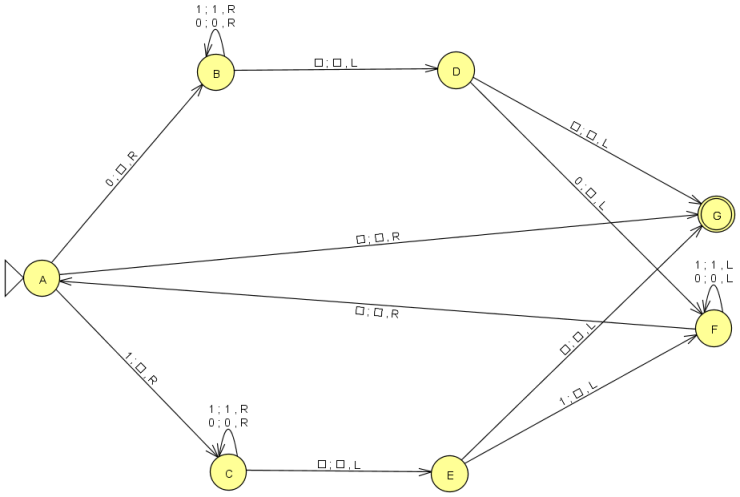
Figura 9: Reglas de finalización, control del personaje y extras.

Nos gustaría mencionar también que, para esta máquina, hemos hecho una ligera modificación para que pueda computar más de una sola palabra. Esta modificación consiste de unas cuantas reglas que podrán

ser manipuladas por el personaje del jugador para indicar cuándo se quiere salir del nivel y cuando se quiere seguir usando la máquina tras haber acabado el cómputo anterior.

2.2.2. Palindromica?(x)

Crearemos un autómata de Turing que acepte todas las palabras sobre el alfabeto $\Sigma = \{0, 1\}$ que sean palindrómicas. El lenguaje de este autómata lo podemos definir como $L(M) = \{x \in \Sigma^* : x = x^R\}$ y el autómata de Turing que usaremos será $M = (\{A, B, C, D, E, F, G\}, \Sigma, \Gamma, \delta, A, \{G\})$:



(a) Esquema

δ	□	0	1
->A	(G,□,R)	(B,□,R)	(C,□,R)
B	(D,□,L)	(B,0,R)	(B,1,R)
C	(E,□,L)	(C,0,R)	(C,1,R)
D	(G,□,L)	(F,□,L)	-
E	(G,□,L)	-	(F,□,L)
F	(A,□,R)	(F,0,L)	(F,1,L)
*G	-	-	-

(b) Función de transición

Figura 10: Autómata de Turing que verifica si una palabra es palindrómica.

Como es un autómata, debemos aplicar las reglas mencionadas en la generalización para autómatas de Turing. Dejando eso de lado, la implementación es la misma. Primero reservamos un objeto para cada símbolo $s \in (\Sigma \cup \Gamma)$ y tres objetos para cada estado $O \in Q$. Acto seguido aplicamos las reglas de ocultación, limpieza y movimiento:

Figura 11: Reglas de movimiento, limpieza y ocultación

Luego usaremos la función de transición para crear las reglas de lectura y escritura. Debemos tomar en cuenta las reglas especiales para cuando la palabra debe ser rechazada tras haber leído un símbolo inválido.

(a) Lectura y dirección

(b) Escritura

(c) De M a L y Generador de E

Figura 12: Reglas que controlan la lectura y escritura de “Palindromica?(x)”.

Por último, añadiremos las reglas que se encargan de la finalización del cómputo:

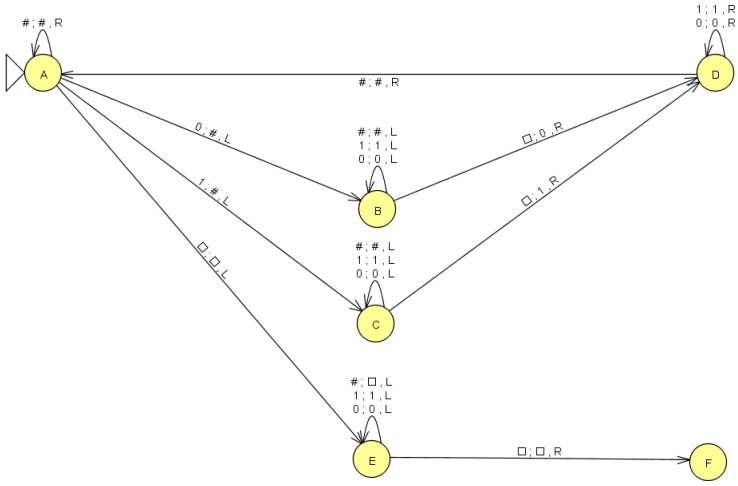
Figura 13: Reglas de finalización, control del personaje y extras

2.2.3. Invertir(x)

Crearemos una máquina de Turing que compute la palabra siguiente de otra dentro del alfabeto especificado anteriormente. La función de esta máquina la podemos definir inductivamente sabiendo que $\forall s \in \Sigma,\ \forall w \in \Sigma^*$:

$$\varphi_M(x) \cong x^R = invertir(x) = \begin{cases} \varepsilon, & \text{si } x = \varepsilon \\ s \cdot invertir(w), & \text{si } x = w \cdot s \end{cases}$$

La máquina de Turing que usaremos será $M = (\{A, B, C, D, E, F\}, \Sigma, \Gamma, \delta, A)$, pero esta vez nuestro alfabeto de cinta será $\Gamma = \{\square, \#\}$:



(a) Esquema

δ	□	#	0	1
->A	(E,□,L)	(A,#,R)	(B,#,L)	(C,#,L)
B	(D,0,R)	(B,#,L)	(B,0,L)	(B,1,L)
C	(D,1,R)	(C,#,L)	(C,0,L)	(C,1,L)
D	-	(A,#,R)	(D,0,R)	(D,1,R)
E	(F,□,R)	(E,□,L)	(E,0,L)	(E,1,L)
F	-	-	-	-

(b) Función de transición

Figura 14: Máquina de Turing que cómputa la palabra inversa a otra.

Volvemos a aplicar el método. Para cada símbolo $s \in (\Sigma \cup \Gamma)$ reservaremos un objeto dentro del juego y tres objetos para cada estado $O \in Q$. Como "Baba Is You" no tiene ningún sprite similar al símbolo "#", lo implementaremos usando la imagen de una pequeña estrella. Tomando eso en cuenta, pasamos a implementar las reglas de ocultación, limpieza y movimiento.

Figura 15: Reglas de ocultación, limpieza y movimiento.

Procedemos a añadir las reglas de lectura y escritura usando la función de transición:

(a) Lectura y dirección

(b) Escritura

(c) De M a L y Generador de E

Figura 16: Reglas que controlan la lectura y escritura de “Invertir(x)”.

Por último, añadimos las reglas de finalización y control de personaje.

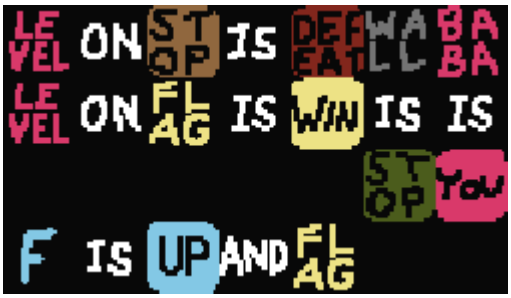


Figura 17: Reglas de finalización, control del personaje y extras.

Ahora que hemos terminado la implementación de los ejemplos, podemos visitar los anexos para encontrar el documento donde se encuentra una captura del nivel final o los links para ver a las máquinas realizar los cálculos.

Conclusión

Hemos logrado mostrar que en "Baba Is You" se pueden implementar máquinas y autómatas de Turing. Al ser el primero uno de los modelos de cómputo más potente que podemos crear, gracias a la tesis de Church-Turing, podemos decir que "Baba Is You" tiene tanta potencia como un lenguaje de programación ya que puede generar algoritmos; sin embargo, esto no lo hace un lenguaje de programación como tal. Si decimos que un lenguaje de programación es un lenguaje que le proporciona al usuario la capacidad de escribir una serie de instrucciones de manera que éste pueda generar algoritmos, entonces "Baba Is You" sí que podríamos decir que es un lenguaje de programación.

Para elaborar este proyecto, lo dividimos en tres fases distintas. Mencionaremos el tiempo aproximado que nos llevó realizar cada fase:

- Análisis de la implementación: 5 horas. Durante esta fase buscamos documentación e información sobre el juego y métodos para implementar una máquina de Turing dentro de él.
- Diseño, construcción y pruebas: 7 horas. Durante esta fase diseñamos, probamos y construimos las máquinas y autómatas de Turing tanto con el juego como con otros programas para verificar que todo estuviera bien.
- Redacción del informe: 21 horas. La fase donde redactamos el informe final. Consideramos que la ésta fue la parte más difícil del proyecto debido a los problemas que presentó organizar las ideas y redactarlas con cohesión.

Durante este proyecto hemos interiorizado más los conceptos de teoría de computabilidad expuestos anteriormente. Además de haber aprendido cómo utilizar un videojuego para crear máquinas de Turing dentro de él y, por ende, crear algoritmos. A pesar de las limitaciones que presenta "Baba Is You", podemos ver que este videojuego tiene más formas de entretener que las que el desarrollador había imaginado.

Referencias

Copeland, B. J. (2019). The Church-Turing Thesis. En E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Spring 2019). Metaphysics Research Lab, Stanford University. Recuperado desde https://plato.stanford.edu/archives/spr2019/entries/church-turing/

De Mol, L. (2019). Turing Machines. En E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Winter 2019). Metaphysics Research Lab, Stanford University. Recuperado desde https://plato.stanford.edu/archives/win2019/entries/turing-machine/

Level Editor. (2019). Recuperado desde https://babaiswiki.fandom.com/wiki/Level_Editor

N.M. Nagornyi, S. M. (2013). Turing Machine. Recuperado desde https://www.encyclopediaofmath.org/index.php/Turing_machine

Nouns. (2019). Recuperado desde https://babaiswiki.fandom.com/wiki/Category:Nouns

Operators. (2019). Recuperado desde https://babaiswiki.fandom.com/wiki/Category:Operators

Properties. (2019). Recuperado desde https://babaiswiki.fandom.com/wiki/Category:Properties

Sánchez, A. (2019). Máquinas de turing y Computabilidad. Recuperado desde https://egela.ehu.eus/pluginfile.php/2263295/mod_resource/content/4/Computabilidad/TR-Computabilidad2019.pdf

Silverhawke. (2019a). [Baba Is You] Enabling Custom Levels and Level Editor. Recuperado desde https://youtu.be/YP-56DFoxw0

Silverhawke. (2019b). [Baba Is You] Turing machine custom level. Recuperado desde https://youtu.be/hsXpLx4soQY

Teikari, A. (2019). Baba Is You. Recuperado desde https://hempuli.com/baba/

Unofficial Baba Is You Wiki. (2019). Recuperado desde https://babaiswiki.fandom.com/wiki/Baba_Is_You_Wiki

Apéndices

Enlaces externos

Durante el desarrollo de este proyecto, se crearon dos máquinas de Turing y un autómata de Turing. Se ha decidido subir vídeos en los cuales se observa el funcionamiento de cada una de estas máquinas.

- Siguiente(x): Para este máquina de Turing se usó un rango de palabras de su alfabeto. Se empezó con la palabra "1111011" y se terminó con la palabra "00000000". Click aquí para visitar el vídeo.
- Palindromica?(x) - Win: Para este autómata de Turing se utilizó la palabra "01011010". En este caso la palabra es palindrómica. Click aquí para visitar el vídeo.
- Palindromica?(x) - Defeat: Para este autómata de Turing se utilizó la palabra "01001000". En este caso la palabra no es palindrómica. Click aquí para visitar el vídeo.
- Invertir(x): Para esta máquina de Turing se usó la palabra "110110". Click aquí para visitar el vídeo.

Documentos

Se adjunta también un documento con la tabla de reglas y una captura del nivel de cada una de las máquinas implementadas. Este documento se llama "Reglas y Niveles.pdf" y se proporciona con el fin de permitirle al lector observar las reglas de cada nivel con detenimiento.