
# Proyecto Final
# Laberinto

**Alumnos:**

Salvador Orozco Villalever – A07104218
Guillermo Garduño García – A01322809
José Antonio Sánchez Sayago - A01327909

**Profesor:** Dr. Iván Olmos Pineda

**Materia:** Proyecto de desarrollo de videojuegos

**Fecha:** 5 de diciembre de 2016

# Proyecto Final
# Laberinto

## Antecedentes

## Desarrollo

1. Diseño en el motor gráfico seleccionado
2. Diseño Lógico del juego
3. Algoritmo de Inteligencia Artificial
4. Función de evaluación
5. Horizonte limitado
6. Pruebas de escritorio realizadas
7. Resultados

1. Diseño en el motor gráfico seleccionado (GTGE)

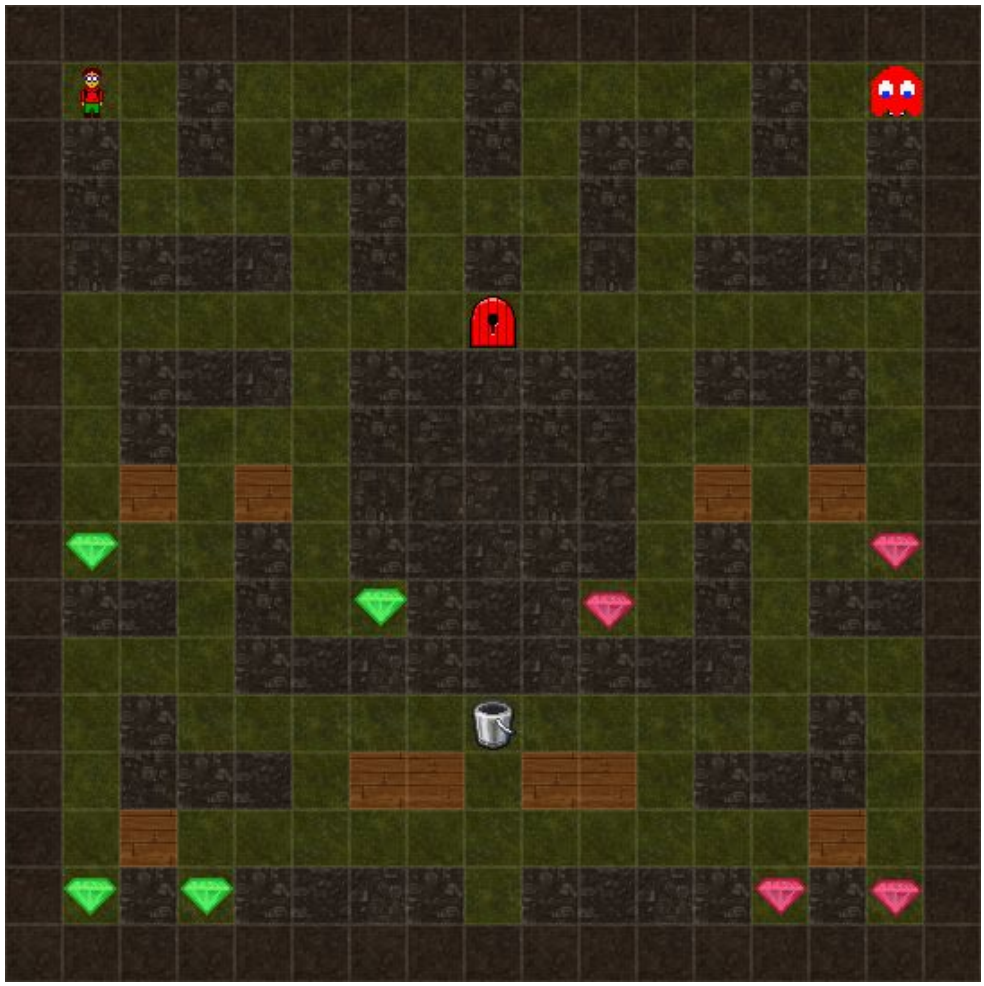

*Imagen 1.1. Diseño del primer nivel*

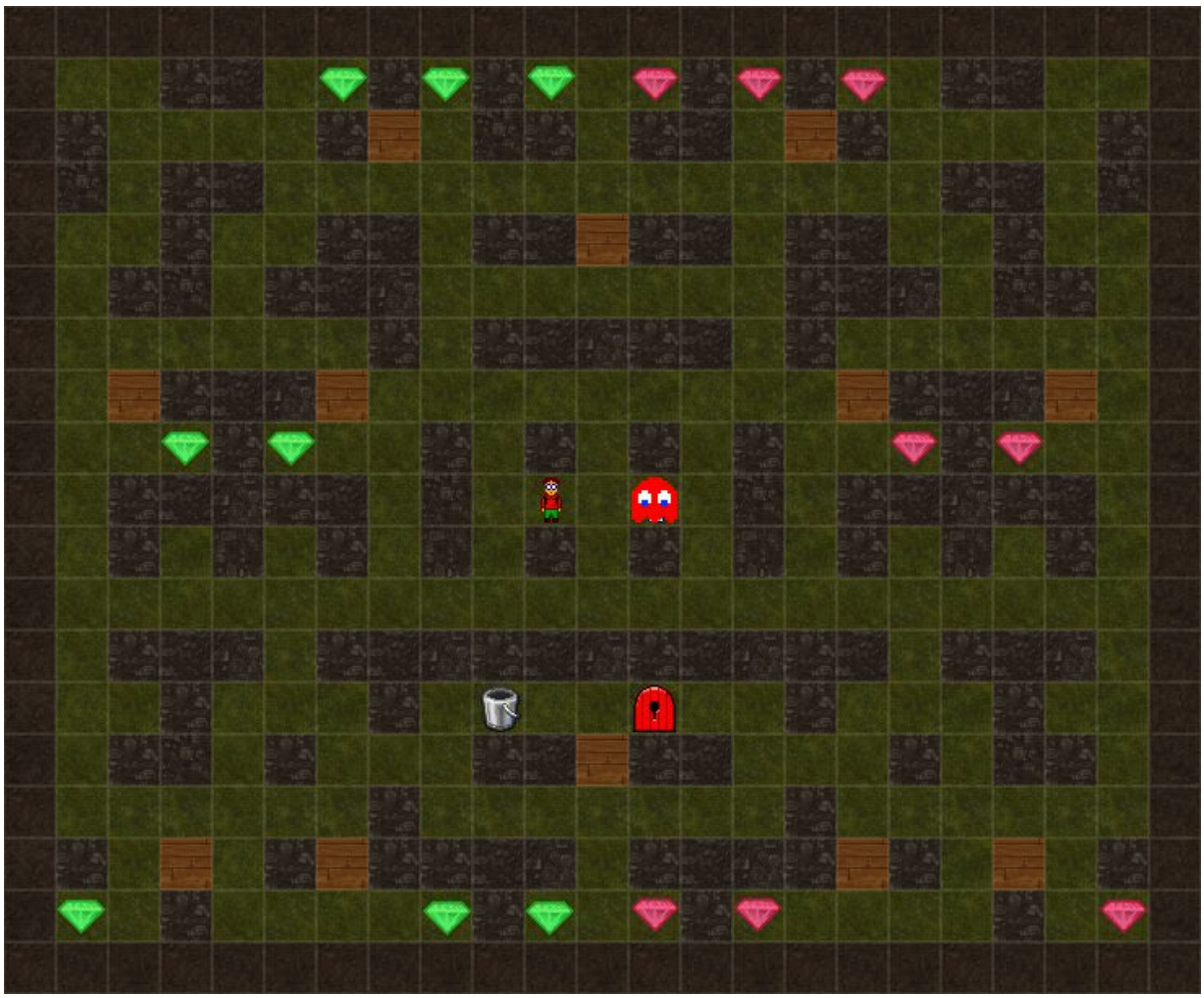

*Imagen 1.2. Diseño del segundo nivel*

## 2. Diseño lógico del juego

Se siguió un desarrollo orientado a objetos. Se escribieron cinco clases que se describen a continuación:

A. **Agente:** clase proporcionada por el profesor

B. **colisionAgentes:** clase proporcionada por el profesor

C. **Main:** es la clase principal, en la cual está el control del videojuego

   a. Los métodos utilizados para la implementación de la inteligencia artificial de nuestro enemigo son los siguientes:

i. **getMovementsR2():** se trata de un método que recibe un nodo inicial a partir del cual se realiza una ejecución del algoritmo A* con el horizonte limitado previamente establecido según el nivel del juego en el cual nos encontremos.

ii. **getMoves():** obtiene las direcciones de los movimientos del enemigo basándose en el arreglo "closed".

iii. **getPath():** obtiene desde el arreglo "closed" el camino para el enemigo; toma decisiones basándose en los índices fijados como índices de padre de cada nodo.

iv. **randomSetTotCoinsPositions():** coloca los diamantes en posiciones aleatorias en el tablero, cuidando que no haya dos diamantes en la misma posición y que la distancia mínima entre diamantes fijada previamente se cumpla entre cualquier par de los mismos.

v. **nodeIsInClosed():** determina si un nodo está presente en closed, de forma que se eviten ciclos.

vi. **moveR2():** a partir del cálculo realizado por el algoritmo A*, este método mueve al enemigo.

vii. **addChildrenToOpen():** agrega nodos hijos de un nodo determinado x al arreglo open cuidando que ninguno de estos hijos esté ya presente en el arreglo closed.

viii. **changeParent():** modifica el índice del nodo padre, índice determinado por la última posición del arreglo "closed" en cada iteración.

ix. **enemyIsBlocked():** determina si el enemigo puede o no moverse, es decir, si puede o no encontrar un camino hacia su objetivo.

x. **enemyWon():** determina si el enemigo ya ganó, es decir, ya recolectó todos sus diamantes y ya atravesó la puerta antes del jugador controlado por el humano.

xi. **stopEnemy():** detiene al enemigo fijando su velocidad tanto en x como y en cero.

xii. **deleteIntelligence():** hace que la inteligencia para los movimientos del enemigo vuelva a empezar desde la casilla en la cual se quedó.

D. **Nodo:** clase utilizada para la inteligencia del agente, es decir, para la implementación del algoritmo A*.

   a. Algunos de los métodos más importantes de esta clase son los siguientes:

      i. **computeManhDist():** calcula la distance de Manhattan entre dos celdas

      ii. **getClosestDiamondCells():** calcula las coordenadas del diamante más cercano al enemigo usando como base la distancia de Manhattan.

      iii. **setValue():** realiza la asignación de la función de evaluación de cada nodo

      iv. **validMove():** determina si se trata de un movimiento válido a partir de una posición determinada

      v. **isDoorOrDiamond():** determina si una celda tiene un diamante o una puerta si es que ya tiene todos los diamantes.

      vi. **coordsAreValid():** determina si una coordenada es válida, es decir, si el enemigo puede llegar a ella con movimientos válidos.

      vii. **computeChildren():** genera los hijos de un nodo determinado, hijos que posteriormente serán procesados por el algoritmo A*.

      viii. **compareTo():** método que compara nodos según el valor de los mismos.

E. **Rock:** clase utilizada para la implementación de obstáculos de tipo roca.

   a. Algunos de los métodos más importantes de esta clase son los siguientes:

      i. **setInitialTime():** fija el tiempo en el cual la roca fue colocada en el mapa. Esto es importante porque las rocas desaparecen después de una cantidad previamente determinada de segundos.

## 3. Algoritmo de Inteligencia Artificial

El algoritmo de inteligencia artificial que utilizamos para el proyecto es el A*. Algoritmo utilizado para encontrar el camino más corto del origen al destino, además es un algoritmo completo, es decir, en caso de que exista una solución, siempre la encontrará. A* hace uso de una función de evaluación $f(n) = g(n) + h'(n)$, $h'(n)$ es un valor heurístico del nodo que será evaluado desde el nodo actual $(n)$, y $g(n)$, representa el costo real del camino que se recorrió para llegar a dicho nodo, *n.*

A* también ocupa 2 arreglos auxiliares, *open* y *close*, dentro del arreglo *open* se guardan todos los "hijos" generados, y el arreglo *closed* guarda los mejores nodos según la función de evaluación y que no hayan sido guardados previamente. Aunado a esto nosotros creamos un arreglo llamado *solutionNodes* donde guardamos el path completo que debe seguir el agente para llegar a cierto punto.

Enfocados en nuestro juego, la idea de general del algoritmo de inteligencia artificial permite al agente inteligente encontrar un camino al diamante más cercano. Este proceso de búsqueda se repite al momento en el que el agente llega al final de su camino, se elimina toda la inteligencia para no gastar la memoria de la computadora.

## 4. Función de evaluación

La heurística que ocupamos para la implementar el algoritmo A* es la *distancia de Manhattan,* una métrica en la cual la distancia entre dos puntos es la suma de las diferencias de sus coordenadas.

## 5. Horizonte limitado

Para el algoritmo A* definimos un horizonte limitado según el nivel del juego en el cual nos encontremos. Para el nivel 1, consideramos adecuado un horizonte de 30, mientras que para el nivel dos consideramos adecuado un horizonte de 45 debido al cambio en las dimensiones del mapa del segundo nivel.

## 6. Pruebas de escritorio realizadas

Para probar que nuestro juego es completamente funcional, realizamos una serie de pruebas que se describirán a continuación:

**Prueba 1:** dejarse perder en el primer nivel
**Prueba 2:** dejarse perder en el segundo nivel
**Prueba 3:** bloquear un camino del enemigo para que realice el recálculo de su trayectoria
**Prueba 4:** bloquear por completo al enemigo, es decir, no permitir que se mueva

**Prueba 5:** ganar el juego
**Prueba 6:** ganar el juego habiendo perdido al menos una vida

Todas fueron satisfactorias.

# 7. Resultados

Como puede observarse en las imágenes anteriormente presentadas, el videojuego funciona correctamente y además tiene todas las secciones requeridas por la rúbrica del proyecto. Por otro lado, se trata de un videojuego completamente jugable y divertido, que además de ser posible ganar al mismo tiempo es interesante y retador para el jugador, de manera que éste no se aburra.


*Imagen 7.1 Presentación del juego.*


*Imagen 7.2 Presentación Nivel 1*


*Imagen 7.3 Nivel 1*


*Imagen 7.4 Presentación Nivel 2*


*Imagen 7.5 Nivel 2*

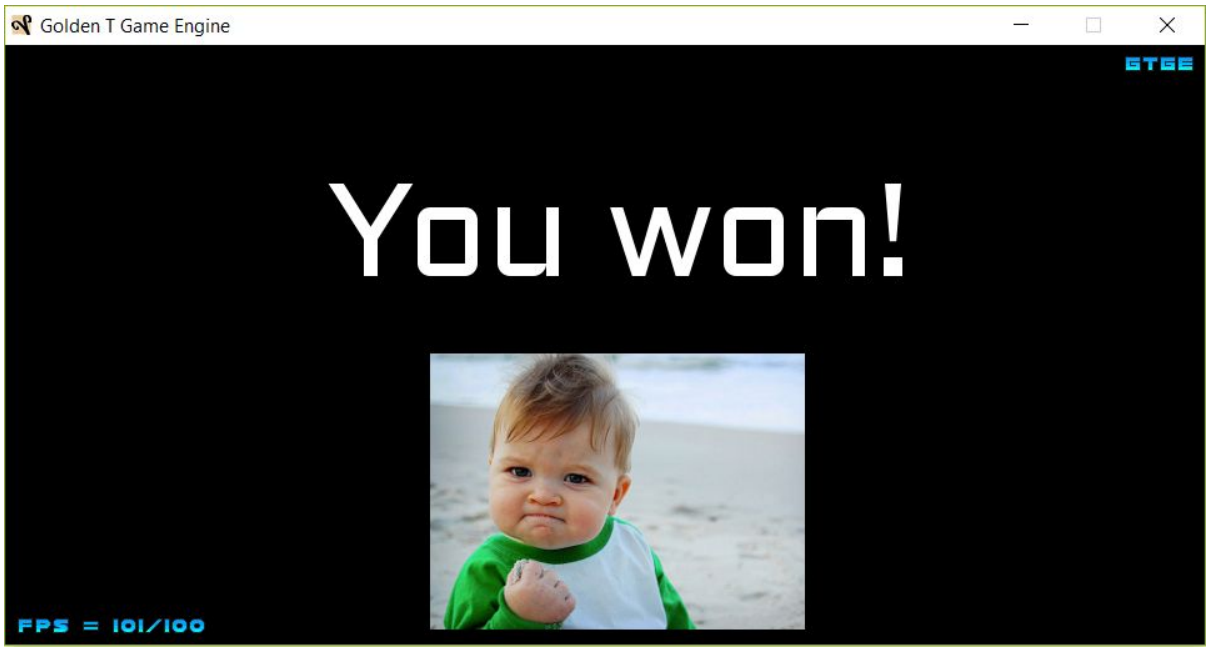


*Imagen 7.6 Mensaje de victoria*


*Imagen 7.7 Mensaje de derrota*

## Conclusiones

Consideramos que este fue un proyecto muy complicado, muy retador, pero súper interesante. Este proyecto nos permitió desarrollar habilidades de comunicación, trabajo en equipo e ingeniería de software de manera que la creación del programa fuera satisfactoria. Además, creemos que nuestras habilidades de resolución de problemas se incrementaron pues a

medida que fuimos diseñando e implementando el código de la solución a cada subproblema del proyecto nos enfrentamos a más subproblemas y más retos de los que imaginábamos.

Creemos que los videojuegos con inteligencia artificial son muy interesantes y divertidos, pues suponen un verdadero reto al usuario. Al programar videojuegos nos encontramos con problemas ya que hay que considerar todos los casos, todos los escenarios posibles dentro del ambiente del juego, tanto en la lógica como en los gráficos. Es precisamente este profundo análisis que debe realizarse lo que dificulta el desarrollo del juego, pero que al mismo tiempo nos impulsa a ser más creativos para poder resolver estos problemas.

Si bien nos tomó mucho tiempo desarrollar esto, si bien estuvimos varias horas súper estresados porque no lográbamos resolver los problemas que tenía el programa o porque creíamos que ya los habíamos resuelto pero surgían de nuevo o surgían otros problemas diferentes, creemos que fue una gran experiencia, un proyecto fascinante.

# Bibliografía

GTGE API v0.2.3. (n.d.). Consultado el 4 de diciembre de 2016 en http://goldenstudios.or.id/products/GTGE/docs/