Universidad Católica de la Santísima Concepción
Facultad de Ingeniería

Aplicaciones de la Inteligencia Artificial

Tarea 3 “Búsqueda”

Integrantes: Cristian Vásquez Baeza
Esteban Cadiz Leyton

Docente: Andrés Sánchez C.

Curso: Inteligencia Artificial

Explicación de la segunda heurística

Para mejorar el rendimiento del A*, diseñamos una segunda heurística que parte de la base de la distancia Manhattan y la modifica con un factor de ajuste de 0.3. Este ajuste fue hecho para comprobar qué tanta diferencia podría haber usado la misma heurística, pero escalada a un número menor, pretendíamos que, al tener menos valor, fuera más eficiente en laberintos más largos.

La consideramos como una heurística nueva porque al multiplicar por algún factor, su comportamiento se ajusta y puede llegar a modificar el recorrido de nodos que haga para encontrar el final. Un factor sobre 1 haría que la búsqueda sea más agresiva para encontrar el objetivo, en cambio, por debajo de uno, es una búsqueda más amplia y que tenderá a recorrer más nodos que su contraparte.

Sin embargo, está la posibilidad de que, a pesar de estos factores, la cantidad de nodos y tiempos de ejecución sean los mismos por la naturaleza de los cálculos, llevando a que no haya diferencia real entre estos aún en los laberintos más grandes.

Comparación de ambas heurísticas

En los tiempos de ejecución, ambas heurísticas tienen una cantidad de tiempo prácticamente igual en cualquiera de los laberintos puestos como prueba, y la única variación que puede haber, se presenta incluso comparando los tiempos de distintas sesiones de la misma heurística, por lo que consideramos que no hay ninguna diferencia entre la heurística Manhattan convencional y su versión escalada.

A los nodos les ocurre lo mismo que a los tiempos, siendo irrelevante tanto si usamos la heurística Manhattan o la escalada, donde la variación observada es independiente de la heurística que utilicemos, y, además, en donde se visitan decenas de millones de nodos en los laberintos más grandes, apenas varía en un par de miles entre cada sesión de prueba.

Conclusiones sobre rendimiento

Tras realizar múltiples sesiones de prueba en laberintos de diferentes tamaños y configuraciones, no observamos diferencias relevantes en el rendimiento entre ambas heurísticas, incluso en laberintos grandes. Tanto en el tiempo de ejecución como en la cantidad de nodos expandidos, los resultados fueron casi idénticos para la heurística de Manhattan y la segunda heurística escalada a 0.3.

Esto quiere que el ajuste aplicado a la heurística no tuvo el impacto esperado en la toma de decisiones del algoritmo, como se creyó en un principio. Es posible que el factor de 0.3 no haya sido suficientemente distinto de 1 como para modificar el comportamiento de la búsqueda en una medida significativa.

Otra explicación podría ser que las características de los laberintos evaluados no representen de forma adecuada los escenarios donde una variación en la heurística haga una diferencia notable, aunque como se probó en varios con tanta diferencia de tamaño, se puede concluir que no tuvo impacto alguno.

Migración de lenguaje de programación

Decidimos migrar nuestro código a C++ con el objetivo de mejorar el rendimiento general del algoritmo. C++ es superior a C# en cuanto a la velocidad de muchos algoritmos, lo cual fue el caso del nuestro, que en C# tardaba entre 40 y 50 minutos en los laberintos más grandes, y al migrarlo a C++, solo tarda poco más de un minuto.

Una de las principales ventajas de C++ es su eficiencia en el manejo de memoria. A diferencia de otros lenguajes, C++ permite un control detallado sobre la memoria, lo cual es crucial para algoritmos que trabajan con estructuras de datos grandes. Esto nos permite reducir el procesamiento y optimizar el uso de los recursos, garantizando un rendimiento más eficiente.

Nuestro nuevo lenguaje utilizado compila de forma más rápida y eficiente, puesto que traduce nuestro código inmediatamente al lenguaje de la computadora, a diferencia de C#, el cual, a pesar de tener compatibilidad con múltiples sistemas operativos,

Gracias a todo esto, la migración fue capaz de reducir de forma drástica el tiempo de ejecución, usando también la versión “Relase” de Visual Studio 2022 en lugar de “Debug”, un gran motivo también por el que el rendimiento aumenta tan significativamente, ya que reordena y no añade tantas verificaciones como su contraparte, optimizando su velocidad y uso de recursos de hardware.