Звіт до лабораторних робіт

*“Інтелектуальні системи”*

Зуєва Антона, Пядика Любомира, МІ-4, 2019 р.

# 1. Постановка задач та планування їх розв’язання

*Задача* *пошуку маршрутів в лабіринті на прикладі гри Пакман*

Необхідно створити власний прототип ведення абстрактної гри. Прототип має містити абстрактний клас або інтерфейс, що буде описувати загальну частину (Стан, Початковий стан, Функція визначення нащадка, Перевірка цілі, Вартість шляху).

В якості реалізації необхідно розробити спрощений варіант гри Pac-Man. Задача - в існуючому лабіринті пакмен має знайти шлях до цілі та з'їсти її.

Шлях шукати різними способами використовуючи алгоритми пошуку оптимальних маршрутів та виводити статистику:

- Час витрачений на пошук

- Кількість кроків зроблену під час пошуку

- Кількість оперативної пам'яті, що була витрачена на пошук

Та найголовніше необхідно, щоб графічний інтерфейс був схожий на оригінал.

Алгоритми пошуку повинні бути наступними: DFS (Depth first search – пошук у глибину), BFS (Breadth first search – пошук у ширину), Greedy Algorithm (деяка реалізація жадібного алгоритму), А\* (еврестичний алгоритм пошуку, що подібний до алгоритму Дейкстри, але із скерованим напрямом пошуку), Minimax (алгоритм мінімізації для ворогів та максимізації для гравця набраних монет).

Для реалізації Minimax в лабіринт необхідно додати: Точки. За кожну точку, що пакман з'їв додавати +1 бал. Якщо пакман з'їв всі точки на карті це означає, що він виграв. Привидів. Привиди рухають картою і намагаються з'їсти пакмана. Якщо привид з'їв пакмана – програш. Рівні. З кожним рівнем збільшується складність гри.

Перед початком реалізації було ретельно сплановано хід дій кожного студента. Спершу, одночасно розбиралися із необхідною кодовою базою для розвязання поставленої задачі, а вже опісля разом обдумувались оптимальні реалізації необхідних алгоритмів. Як тільки дійшли до згоди, перейшли до поділу алгоритмів: перші три розроблялися Антоном, а останні два – Любомиром. У третьому пункті можна детальніше почитати про деякі цікавинки, які виникали під час реалізації.

*Задача формування розкладу факультету*

Написати програму, що формує розклад факультету використовуючи генетичні алгоритми.

Вимоги:

- в системі мають бути дисципліни, що мають лекції та практичні

- мають бути обмеження, на аудиторії в яких проводяться зайняття (так припустимо в аудиторії, що містить 15 місць не можна провести лекцію)

- практичні зайняття можуть проводити різні викладачі

- один викладач не може проводити одночасно різні пари

- один студент не може одночасно бути на різних парах

- розклад занять має відповідати стандарту запровадженому на факультеті (початок та кінець пар, навчальні дні)

Не важливо як дані та обмеження потрапляють в систему (інтерфейс наповнення даними та обмеженнями). Головне результат - реалізація алгоритму та складений розклад.

Також переписати програму, що формує розклад факультету розглянувши її як CSP задачу.

Використати евристики:

- Minimum Remaining Values

- степеневу евристику

- евристику з найменш обмежуючим значенням

- forward checking

- constraint propagation

Дослідити як дані евристики впливають на швидкість роботи алгоритму.

Подібно до попередньої задачі, було продумано хід необхідних дій для розвязання. Генечний алгоритм рзроблявся спільними зусиллями, а евристики із поділом: Любомир перші два та аналіз швидкості роботи, а Антон решта три.

# 2. Опис алгоритмів

*DFS – пошук в глибину*. Нехай G = (V, E) - простий зв'язний граф, усі вершини якого позначено попарно різними символами. У процесі пошуку вглиб вершинами графа G надають номери (DFS-номери) та певним способом даних для збереження множин, яку називають стеком. Зі стеку можна вилучити тільки той елемент, котрий було додано до нього останнім: стек працює за принципом "останній прийшов - перший вийшов" (LIFO). Інакше кажучи, додавання й вилучення елементів у стеку відбувається з одного кінця, який називається верхівкою стеку. DFS-номери вершини х позначають DFS(х).

*BFS – пошук в ширину*. Якщо задано граф G = (V, E) та початкову вершину s, алгоритм пошуку в ширину систематично обходить всі досяжні із s вершини. На першому кроці вершина s позначається, як пройдена, а в список додаються всі вершини, досяжні з s без відвідування проміжних вершин. На кожному наступному кроці всі поточні вершини списку відмічаються, як пройдені, а новий список формується із вершин, котрі є ще не пройденими сусідами поточних вершин списку. Для реалізації списку вершин найчастіше використовується черга та принцип FIFO.

*Жадібний алгоритм*. Жадібний алгоритм базується на заданні пріоритетів кожному з можливів напрямів руху, причому рух у протилежному напрямі до поточного має найнижчий пріоритет.

*A\* (A Star).* Алгоритм А\* спершу відвідує ті вершини, які ймовірно ведуть до найкоротшого шляху до мети. Аби розпізнати такі вершини, кожній відомій вершині x співставляється значення f(x), яке дорівнює довжинінайкоротшого шляху від початкової вершини до кінцевої, який пролягає через обрану вершину. Вершини з найменшим значенням f обираються в першу чергу*.*

*Minimax.* Мінімаксний алгоритм рекурсивний алгоритм для вибору наступного кроку для ігор з n гравцями, як правило, двох гравців. Евристична оцінка пов'язана з кожним положенням або станом гри. Це значення обчислюється за допомогою евристичної функції, і це показує, як добре було б для гравця досягти цієї позиції. Потім гравець робить хід, який максимізує мінімальне значення позиції в результаті можливих наступних ходів суперника. Якщо черга А робити хід, то А дає оцінку для кожного зі своїх ходів.

*Генетичний алгоритм.* Можна виділити такі етапи генетичного алгоритму:

- Створення початкової популяції:

- Обчислення функції пристосованості для осіб популяції (оцінювання)

- Повторювання до виконання критерію зупинки алгоритму:

- Вибір індивідів із поточної популяції (селекція)

- Схрещення або/та мутація

- Обчислення функції пристосовуваності для всіх осіб

- Формування нового покоління

*CSP (Constrain Satisfaction Problem) – задача виконання обмежень*. CSP - це математичні проблеми, визначені як сукупність об'єктів, стан яких має задовільняти ряду обмежень. ЗВО предсатвяє сутності проблеми як однорідний набір обмежень, що накладаються на змінні, які розв'язуються методами виконання обмежень. ЗВО є предметом інтенсивних досліджень і в галузі штучного інтелекту, і дослідженні операцій, оскільки закономірності у формулюванні цих задач складають загальну основу для аналізу та вирішення проблем в багатьох неспоріднених областях. ЗВО часто мають високу складність, що вимагає поєднання евристичних та комбінаторних методів пошуку для швидкого вирішення.

*Minimum Remaining Values - евристикою з мінімальною кількістю значень, що залишилися.* Інтуїтивна ідея згідно якої в першу чергу необхідно обирати змінну з найменшою кількістю «допустимих» значень.

*Степенева евристика.* Обираємо змінну, що приймає участь в найбільшій кількості обмежень на інші змінні з не привласненим значенням.

*Least constraining value - евристика з найменш обмежуючим значенням.* Перевага віддається значенню, при якому з розгляду виключається найменша кількість варіантів вибору значень для сусідніх змінних в графі обмежень.

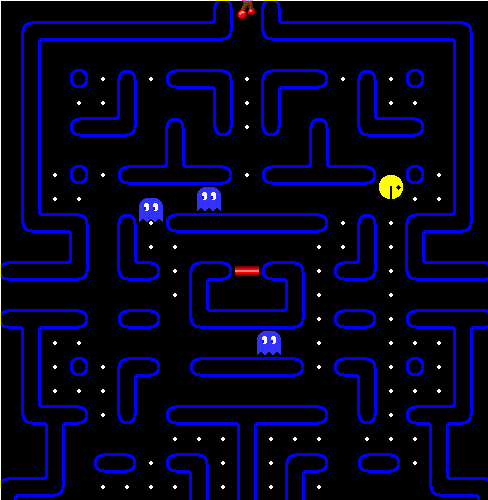
*Forward checking — пошук з поверненнями.* Пошук в глибину в якому кожний раз вибираються значення для однієї змінної і виконується повернення, якщо більше не залишилося допустимих значень, що можна було б привласнити змінній називається пошуком з поверненнями.

*Constraint propagation.* Обмеження поширюються на кожну під змінну.

3. Програмна реалізація

Розглянемо декілька цікавих моментів, результатів, які виникали впродовж реалізації. Більш детальну інформацію у вигляді лістингів коду можна переглянути на GitHub за посиланням <https://github.com/cuberleon/univ/tree/master/it>.

Знімок з екрану під час виконання алгоритму Minimax у грі пакман:



На наступних двох знімках показано проблему, яка виникає із алгоритмом DFS. Проблема полягає у зациклення пакмана рухаючись постіно туди і назад. Якщо карту представити у вигляді графу, і оскільки даний граф не має тупіків і він звязний, то із кожної точки на карті можна дійти до будь-якої іншої, причому рухатись можна у довільному напрямі, який не веде до стіни. Так, як DFS находить перший ліпший маршрут до цілі, то по суті даний алгоритм можна звести до вибирання першого незагородженого напряму із списку пріоритетів. DFS у явному вигляді не можна застосовувати до даної задачі. Згаданий вище жадібний алгоритм розвязує проблему зациклення слідуючи пріоритетам напрямів.

При розвязанні іншими алгоритмами труднощів та проблем не виникало.

Порівняння швикодії алгоритмів:

dfs: path\_len=4, iters\_cnt=4 time\_spent=0.264ms

bfs: path\_len=4, iters\_cnt=6 time\_spent=0.335ms

greedy: path\_len=4, iters\_cnt=4 time\_spent=0.129ms

a\_star: path\_len=3, iters\_cnt=3 time\_spent=0.376ms

minimax: iters\_cnt=1315 score=0.238 time\_spent=35.945ms

Порівняння швидкодії алгоритмів, що розвязують задачу побудови розкладу:

1. req\_cnt = 60, teacher\_cnt = 6, groups\_cnt = 4

func = mrv, time = 127.226ms

func = step\_evr, time = 16.183ms

func = lcv, time = 9683.792ms

func = forward\_checking, time = 4.334ms

func = constraint\_prop, time = 4.189ms

2. req\_cnt = 44, teacher\_cnt = 6, groups\_cnt = 4

func = mrv, time = 77.705ms

func = step\_evr, time = 7.360ms

func = lcv, time = 4088.237ms

func = forward\_checking, time = 2.668ms

func = constraint\_prop, time = 2.522ms

3. req\_cnt = 16, teacher\_cnt = 6, groups\_cnt = 4

func = mrv, time = 15.700ms

func = step\_evr, time = 1.165ms

func = lcv, time = 205.958ms

func = forward\_checking, time = 0.564ms

func = constraint\_prop, time = 0.570ms