**HO CHI MINH UNIVERSITY OF SCIENCE**

**FACULTY OF INFORMATION TECHNOLOGY**



**REPORT: Project 01:**

**Hide and Seek**

By:

22127057 – Đỗ Phan Tuấn Đạt

22127064 – Phạm Thành Đạt

22127123 – Lê Hồ Phi Hoàng

22127131 – Trần Nguyễn Minh Hoàng

TABLE OF CONTENTS

[A. INTRODUCTION 3](#_Toc163066655)

[B. WORKING ENVIRONMENT 4](#_Toc163066656)

[I. Programing language 4](#_Toc163066657)

[II. Code editor 4](#_Toc163066658)

[III. Code management 4](#_Toc163066659)

[C. GENERAL CODE ANALYSIS 5](#_Toc163066660)

[I. Seeker Agent 5](#_Toc163066661)

[1. Code representation 5](#_Toc163066662)

[2. Seeker’s vision 5](#_Toc163066663)

[3. Path-finding using search algorithms 6](#_Toc163066664)

[D. LEVEL-SPECIFIC ANALYSIS 7](#_Toc163066665)

[I. Level 1 7](#_Toc163066666)

[E. REFERENCES 7](#_Toc163066667)

# INTRODUCTION

This project was initialized as part of the course CSC14003 – Introduction to Artificial Intelligence. Its purpose is to help the students learn the basics of creating and implementing artificially intelligent agents through the simple childhood game hide-and-seek.

Note that the agents in this project are in no way capable of learning since all utilized algorithms are search algorithms, not neural networks.

# WORKING ENVIRONMENT

## Programing language

100% Python

## Code editor

Visual Studio Code

## Code management

GitHub

# GENERAL CODE ANALYSIS

## Seeker Agent

### Code representation

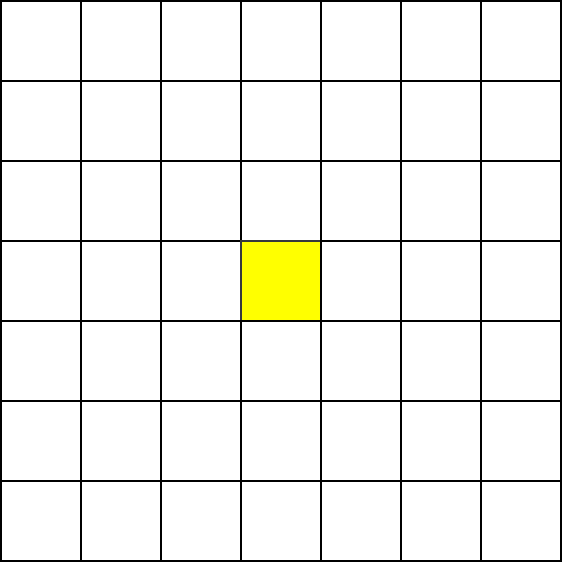
* The seeker agent is represented as the Seeker class, stored in the seeker.py file of each level.
* A Seeker object is constructed as follow:



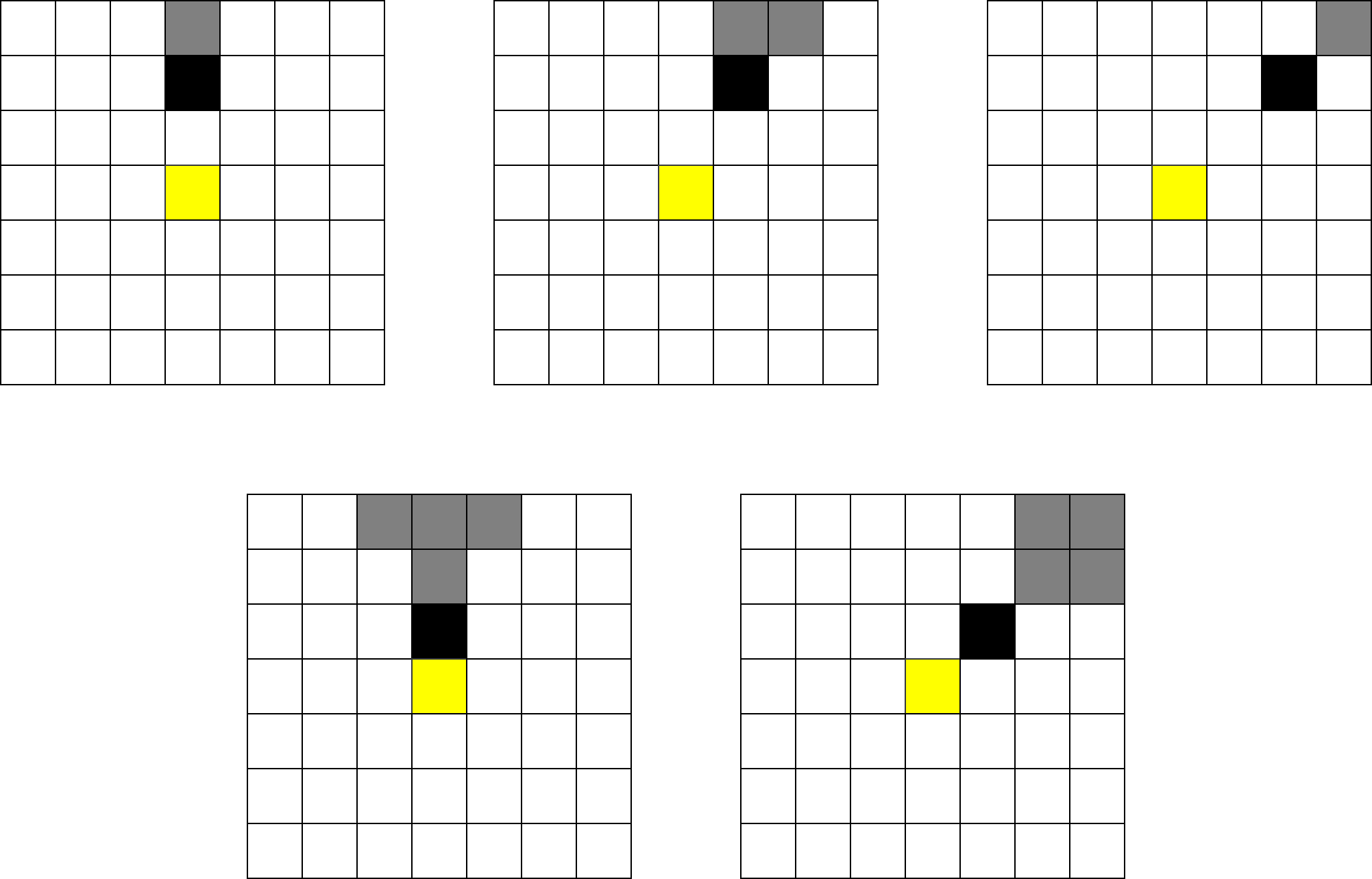
* In case you haven’t realized, this Seeker class also serves a dual purpose as a representation of the puzzle state, hence the *path\_cost*, *heuristic\_cost* and *parent* paremeters used in initialization, as well as the *\_\_lt\_\_* operator implemented further down the source code. This may look a bit odd, but it will come in handy later.

### Seeker’s vision

* As required, the seeker has a vision that ranges 3 tiles away from it in all 8 directions, as illustrated below:



* When there is an obstacle/wall within its range of vision, observable tiles are updated as follow:



(**Black tiles** represents obstacles/walls; **grey tiles** are those that are now unobservable. The same logic applies for other quarters within the range of vision)

* In cases of long walls or multiple obstacles, observable tiles are updated accordingly.

### Path-finding using search algorithms

* Within the Seeker class, 3 search algorithms are implemented:
  + Breath-first search (BFS)
  + Hill climbing search, in which the seeker tries to move towards positions with the highest possible number of observable tiles.
  + A\* search using Manhattan distance between current position and target position as the heuristic function.
* The priorities of each search strategies are as follow:
  + When neither hider or signal is present within observable tiles, traverse the map using Hill climbing search.
  + Upon encountering a local maximum (when all possible moves lead to a reduce in observable ties), switch to BFS to search for another unexplored portion of the map.
  + At any point during traversal, if a signal or hider appears in sight, immediately switch to A\* search to reach the position of said hider/signal.

# LEVEL-SPECIFIC ANALYSIS

## Level 1

* In this level, things are at their most simple form: 1 hider which cannot move, 1 seeker and no time limit; the game ends when the seeker successfully catch the hider (i.e. they occupy the same tile).
* The seeker’s strategy is also at its base form, which will not be the case for later levels.

# REFERENCES

* <https://www.google.com/search?client=firefox-b-d&q=make+startign+mnu+pygame#fpstate=ive&vld=cid:0b2212ca,vid:GMBqjxcKogA,st:0>