# Intelligent Search Algorithms

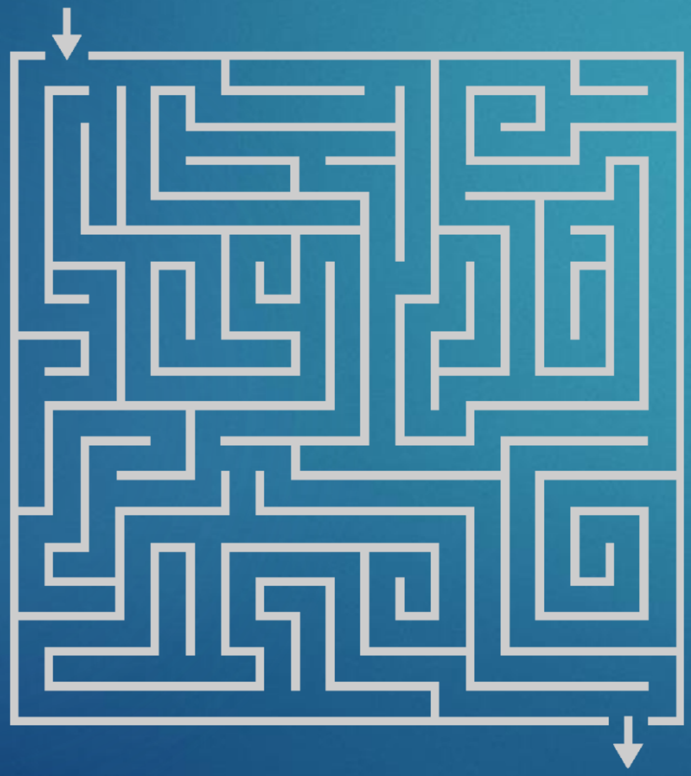
Forth Year - 2022

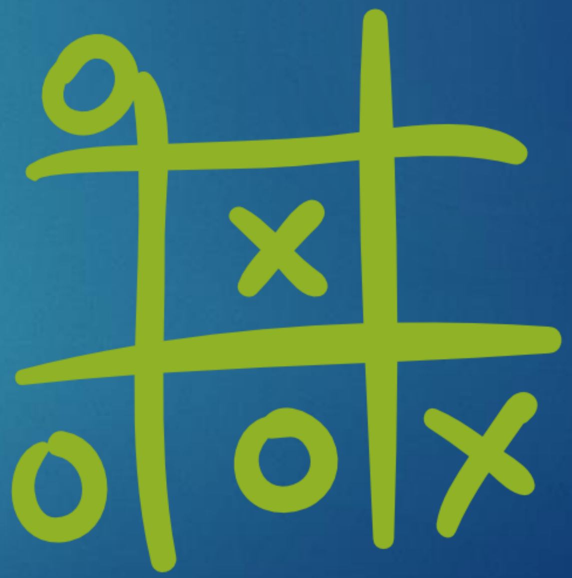
ENG. MOHAMMAD KHAIR KIBABI

# Search Algorithms

## Why we search?

❑ Searching with single agent environment





❑ Searching in multi agent environment

# Search Algorithms Problems

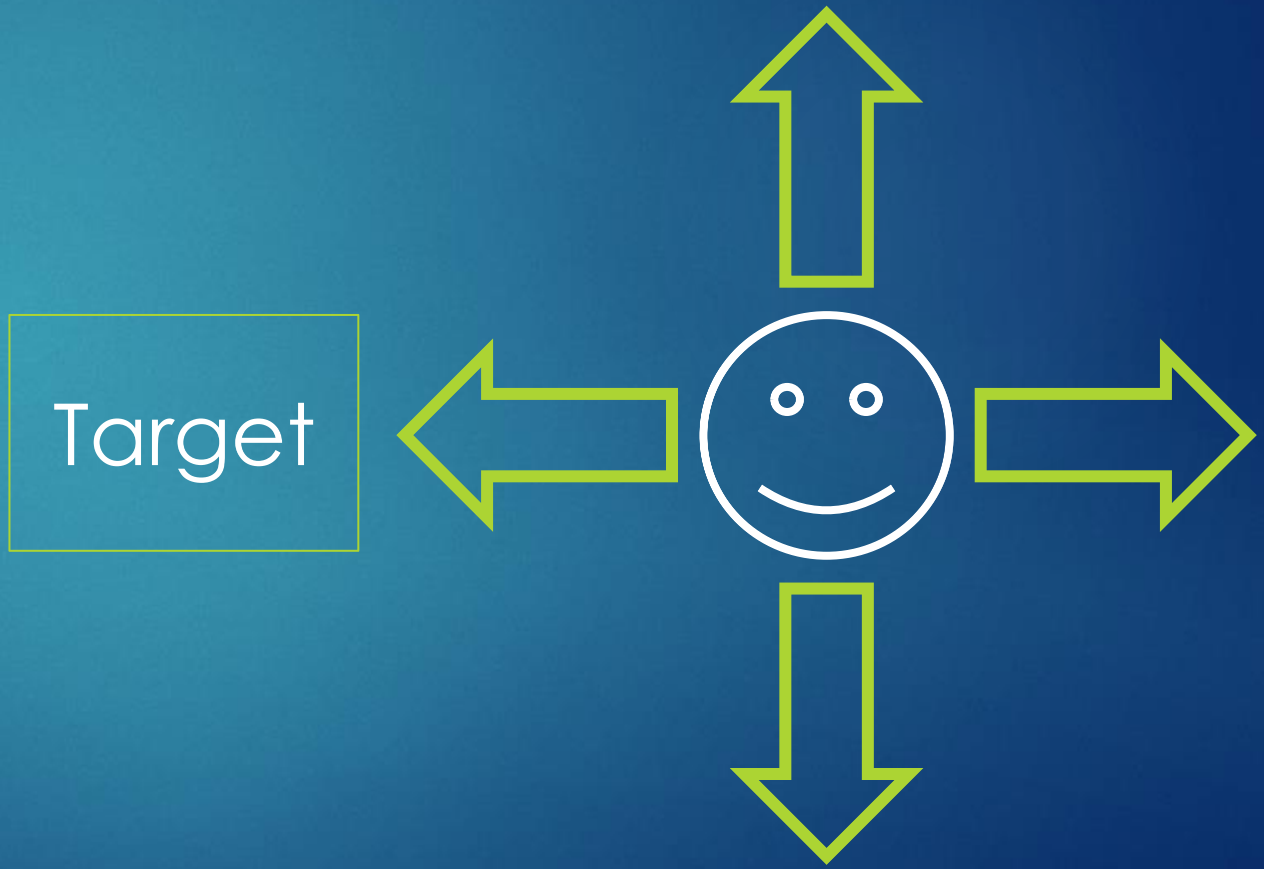
- TIME COMPLEXITY
- SPACE COMPLEXITY
- EXAMPLE : FIND ALL SOLUTIONS IN THE CHESS !!!!

# Intelligent Search Algorithms

- HUMAN DIRECTLY GO

  TO THE TARGET.

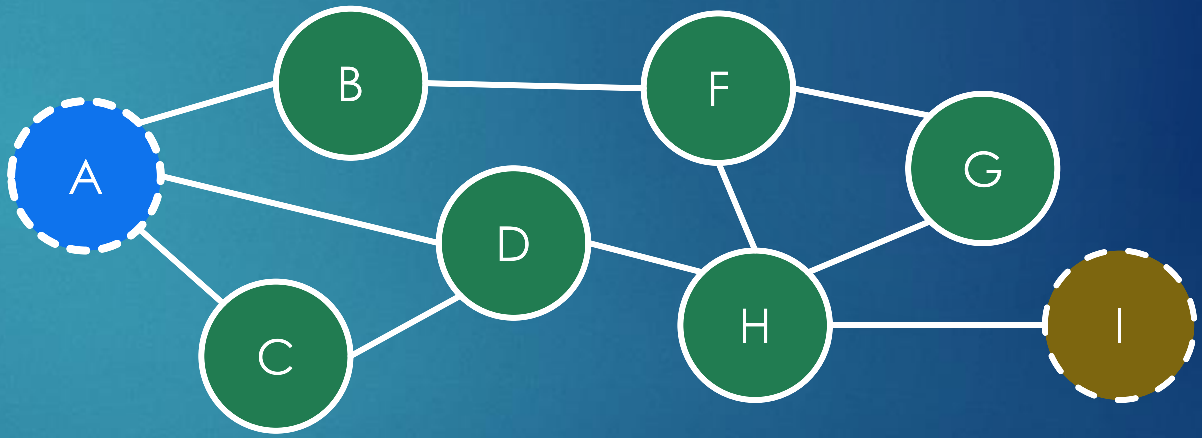
- BUT SOFTWARE ????



# Formulating The Problem

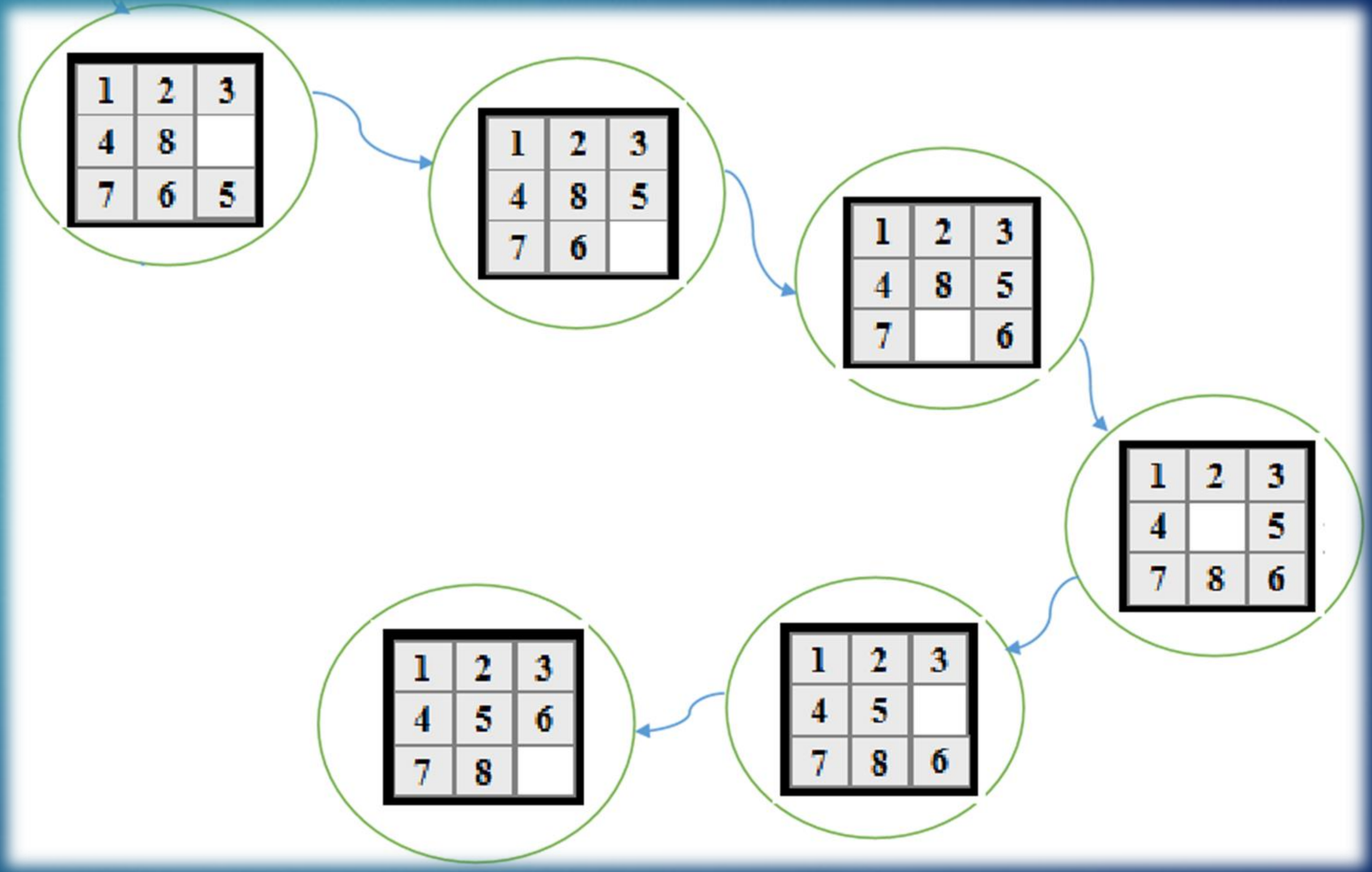
## State Space:

- START STATE
- NODES : STATES
- OPERATORS OR ACTIONS OR EDGES
- GOALS



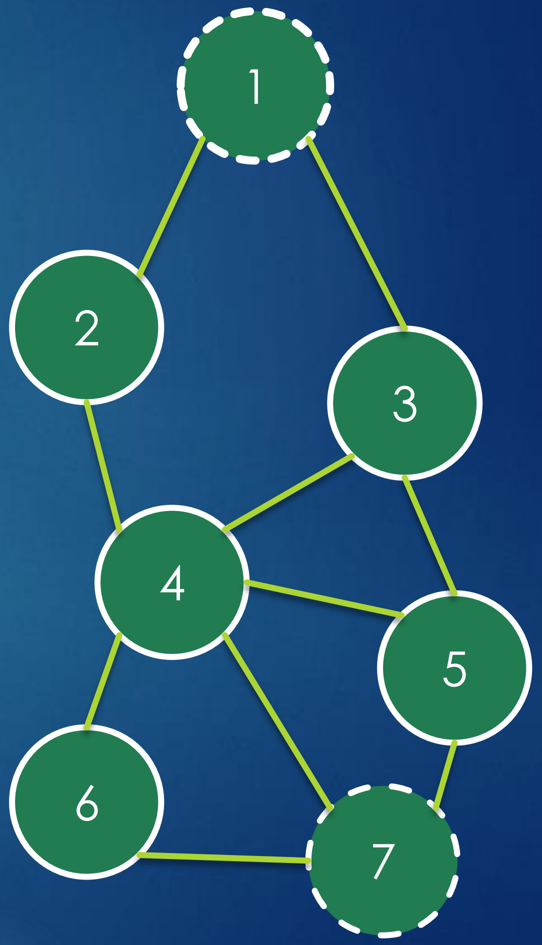
# Formulating The Problem

Nodes (States):



# Search Strategy

- A strategy is defined by picking the order of node expansion.
- Strategy is evaluated among the following dimensions:
  - COMPLETENESS - DOES IT ALWAYS FIND A SOLUTION IF ONE EXISTS?
  - TIME COMPLEXITY- NO. OF NODES GENERATED.
  - SPACE COMPLEXITY- MAX NO. OF NODES IN MEMORY.
  - OPTIMALITY- DOES IT ALWAYS FIND LEAST COST SOLUTION?



# Types of Search Strategies

## 1 - Uninformed Search:

- SEARCH THAT HAS NO INFORMATION ABOUT ITS DOMAIN.
- SEARCH THE NUMBER OF NODES CAN BE EXTREMELY LARGE.
- THE ORDER OF EXPANDING NODES IS ARBITRARY.
- EXAMPLES:
  - Breadth First Search
  - Depth First Search
  - Uniform Cost Search

Blind Search

# Types of Search Strategies

## 2 - Informed Search:

- USE INFORMATION ABOUT THE DOMAIN TO MAKE THE SEARCH PROCESS MORE EFFICIENT.
- INFORM THE SEARCH ABOUT THE DIRECTION TO A GOAL TO GUESS WHICH NEIGHBOR OF A NODE WILL LEAD TO A GOAL.
- EXAMPLES:
  - Hill Climbing
  - A*
  - AO*

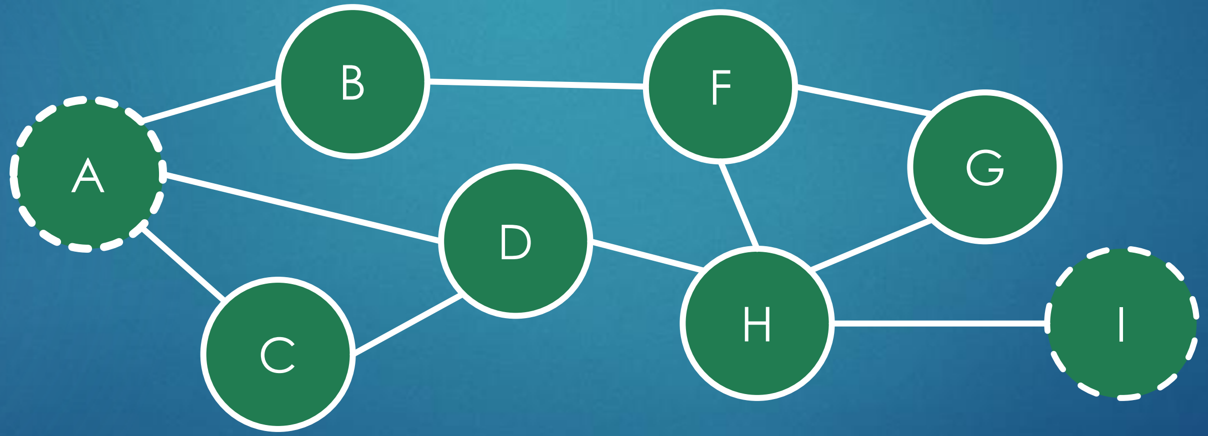
Intelligent Search

# Intelligent Search Algorithms
## Applications

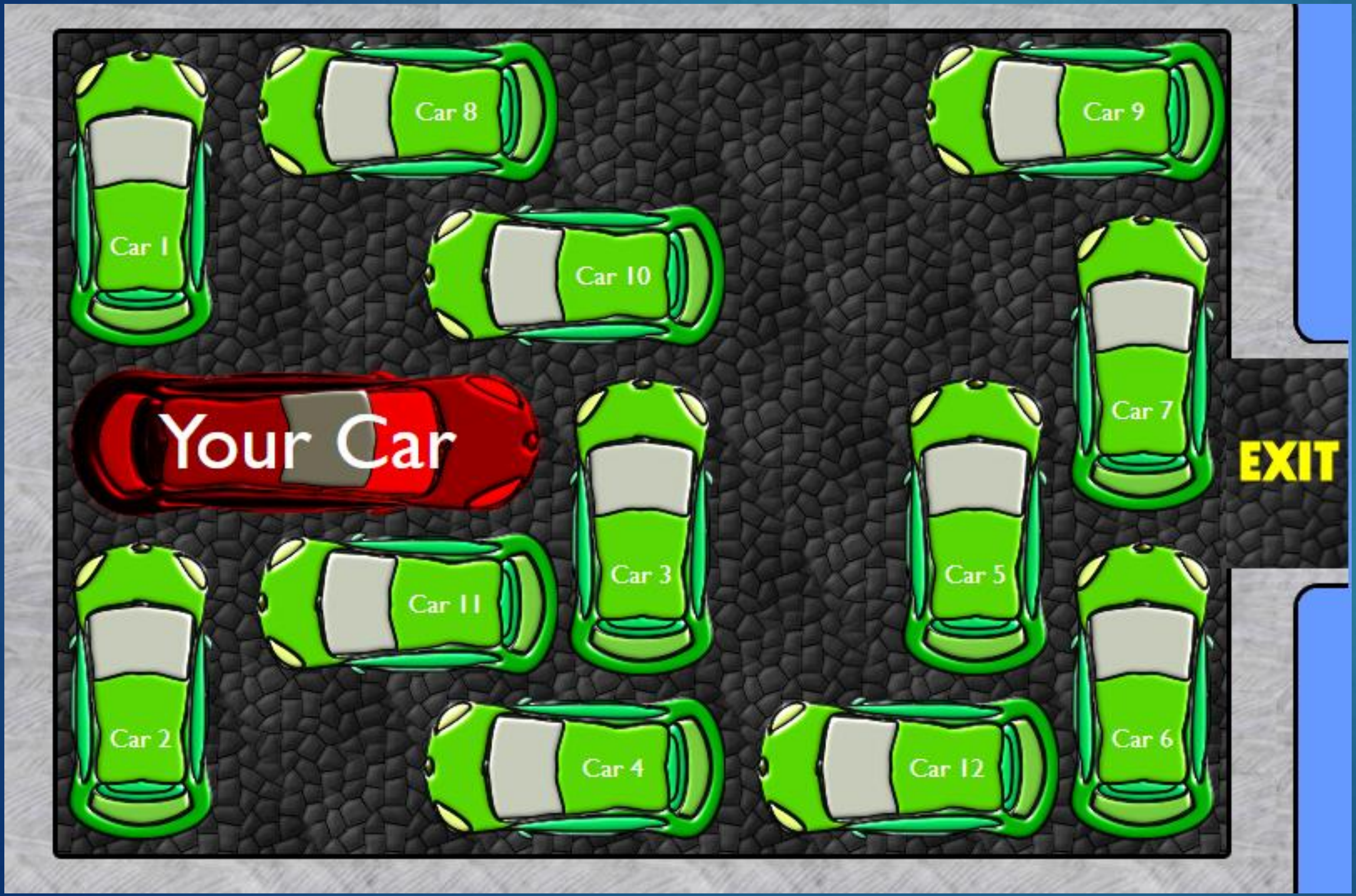
- PROBLEM SOLVING:
  - Puzzles
  - Play games, e.g. chess
  - Scheduling
  - Symbolic integration of mathematical formulas.
- LOGICAL REASONING
  - Prove assertions (theorems) by manipulating a database of facts (like prolog)
- PLANNING:
  - find a sequence of actions to achieve a goal for a robot.
- LANGUAGE:
  - find the best parse of a sentence : e.g. Spelling checker

# Intelligent Search Algorithms

## HOMEWORKS – EXAMPLES

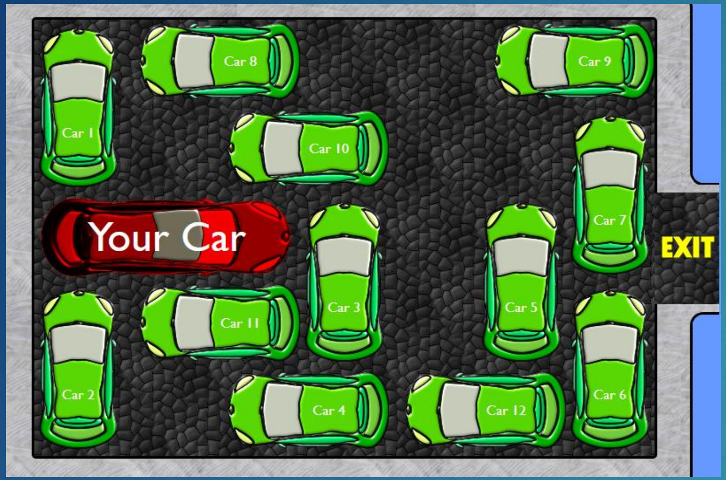


# Car Park Puzzle

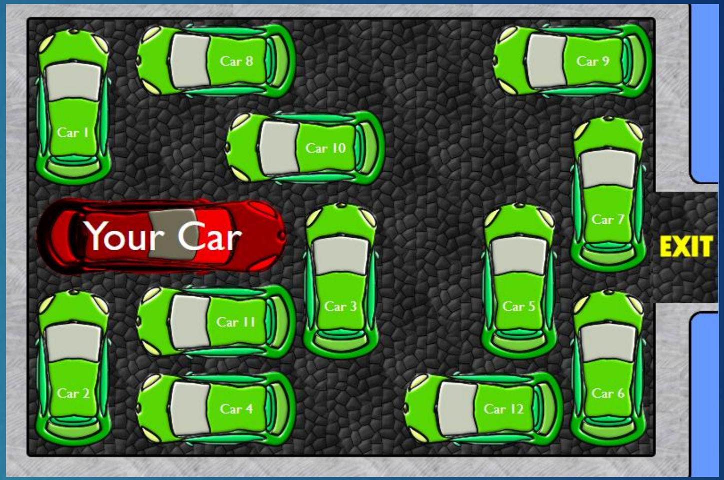



https://www.transum.org/Maths/Investigation/CarPark/Default.asp?Level=1

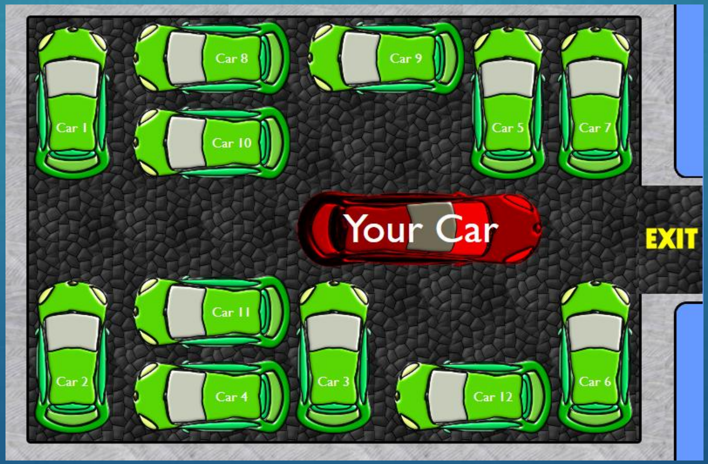
# Game states




Start State




Second State




Before Final State

# Game Structure

Array ?

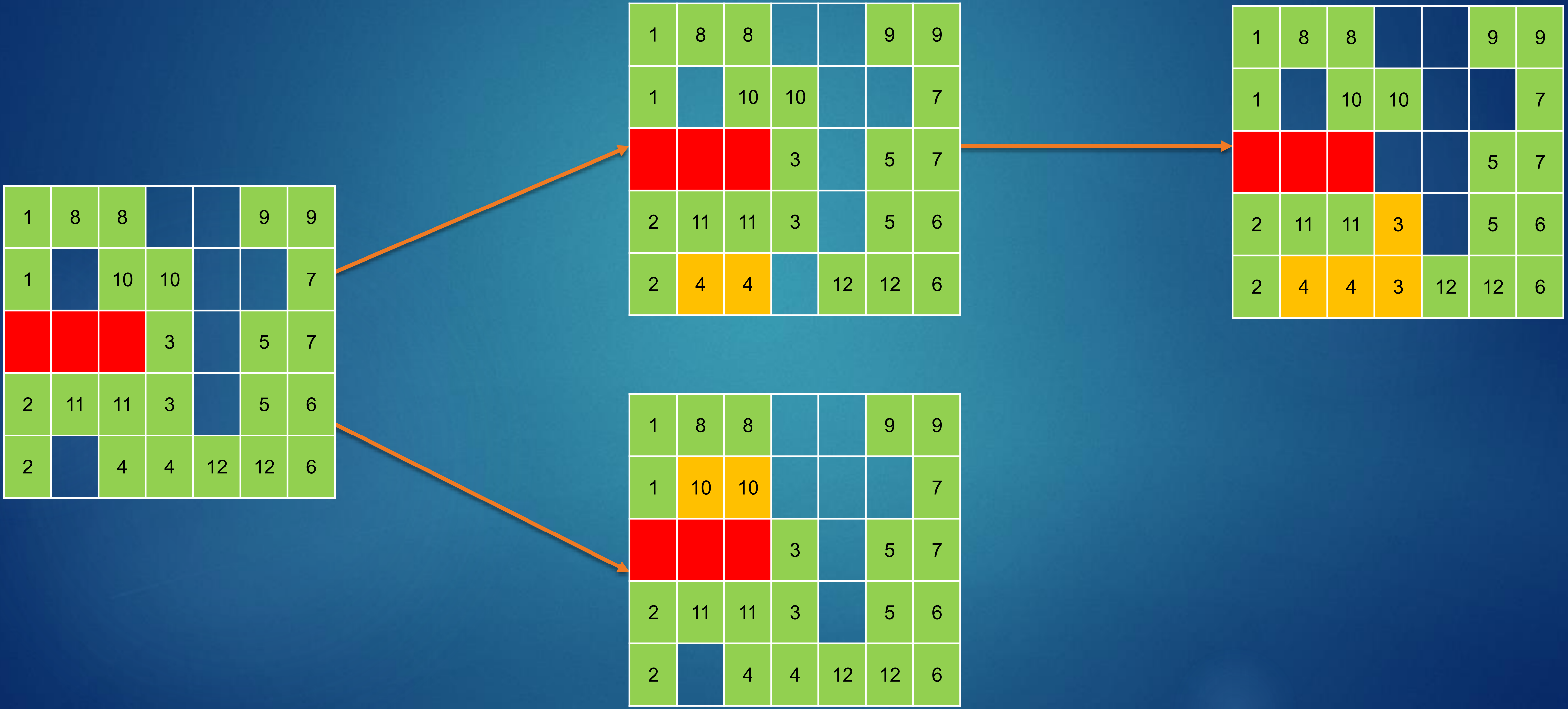
| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 8 | 8 | | | 9 | 9 |
| 1 | | 10 | 10 | | | 7 |
| | | | 3 | | 5 | 7 |
| 2 | 11 | 11 | 3 | | 5 | 6 |
| 2 | | 4 | 4 | 12 | 12 | 6 |

# Piece Movement

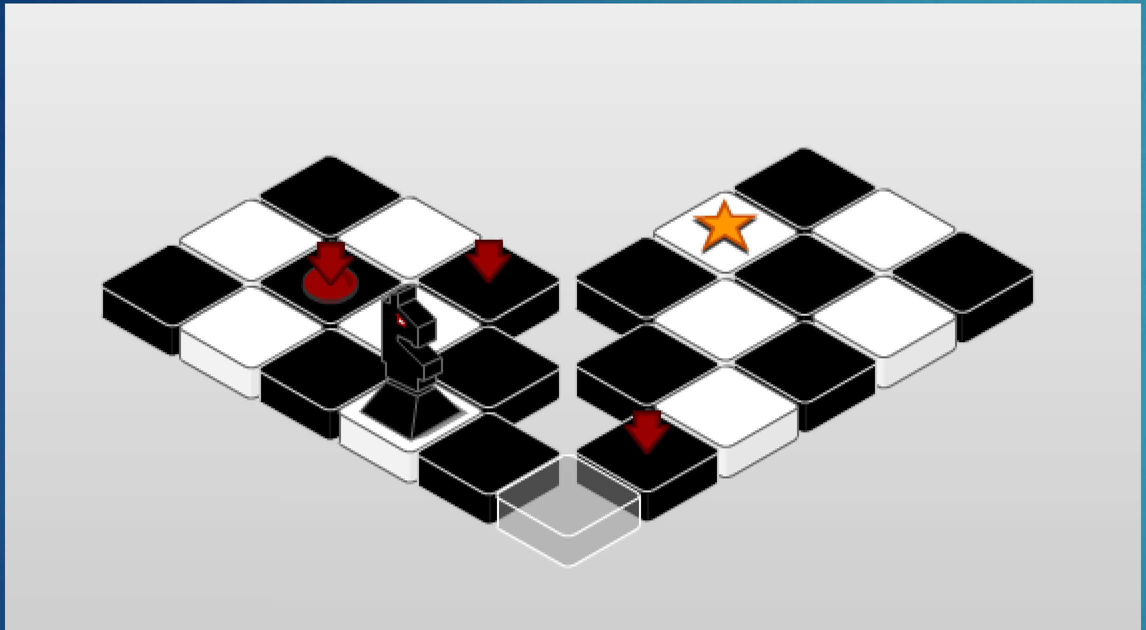
| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 8 | 8 | | | 9 | 9 |
| 1 | | 10 | 10 | | | 7 |
| | | | 3 | | 5 | 7 |
| 2 | 11 | 11 | 3 | | 5 | 6 |
| 2 | | 4 | 4 | 12 | 12 | 6 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 8 | 8 | | | 9 | 9 |
| 1 | | 10 | 10 | | | 7 |
| | | | 3 | | 5 | 7 |
| 2 | 11 | 11 | 3 | | 5 | 6 |
| 2 | 4 | 4 | | 12 | 12 | 6 |

# Print States

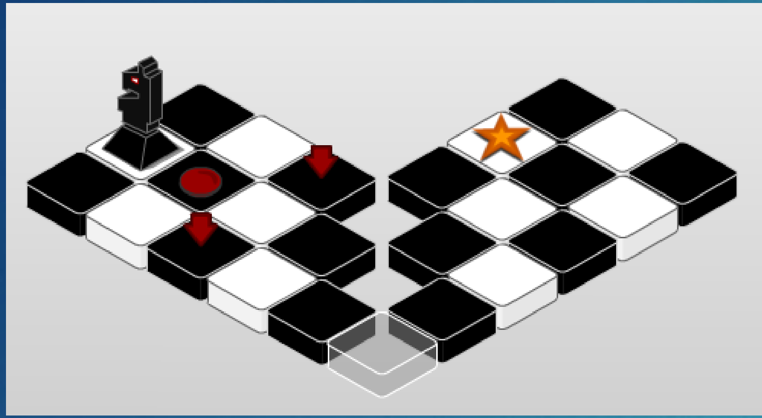


# Black Knight

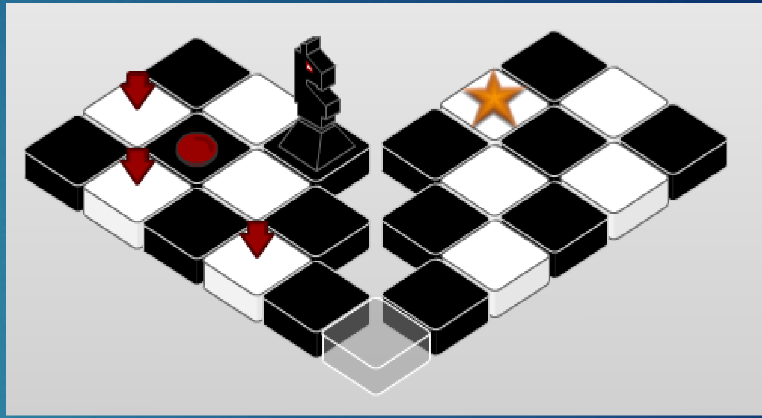


http://www.flonga.com/play/black-knight.htm

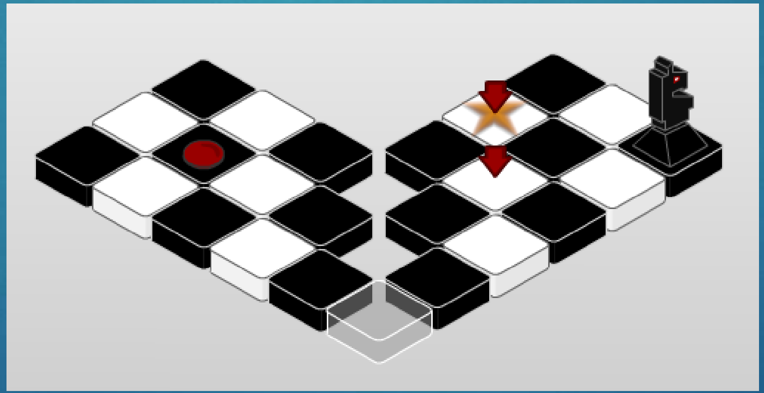
# Game states



Start State



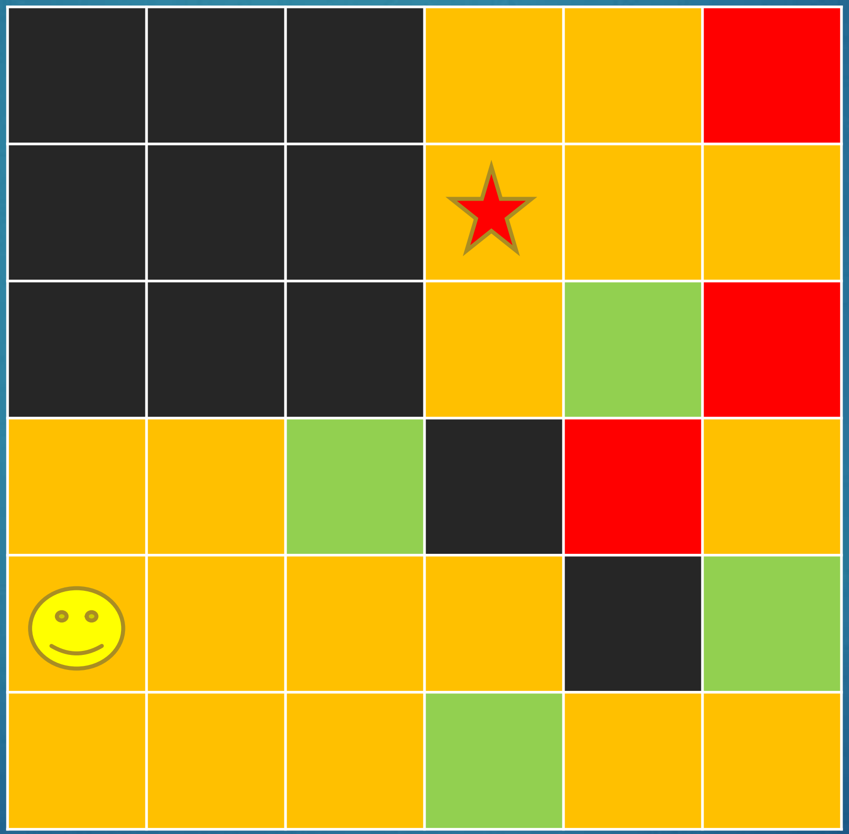
Second State



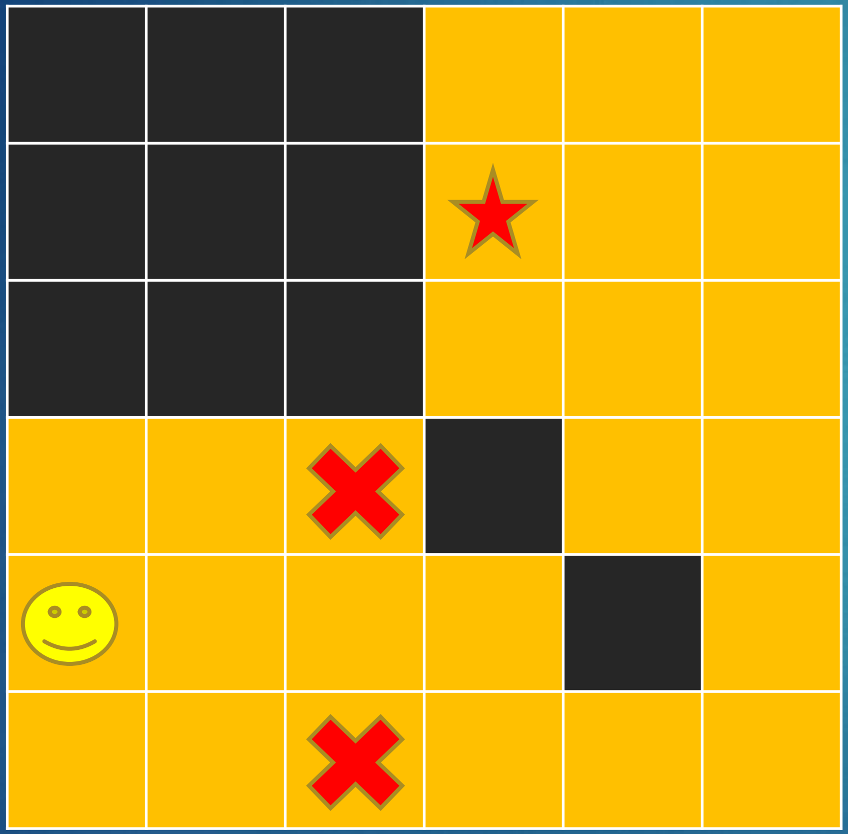
Before Final State

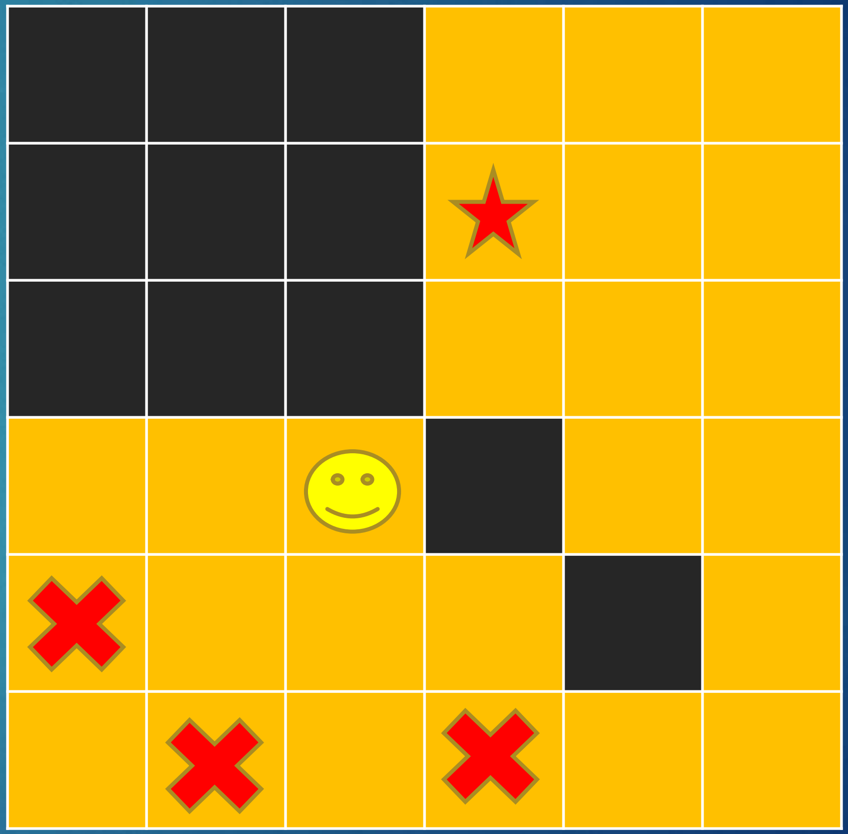
# Game Structure

Array ?

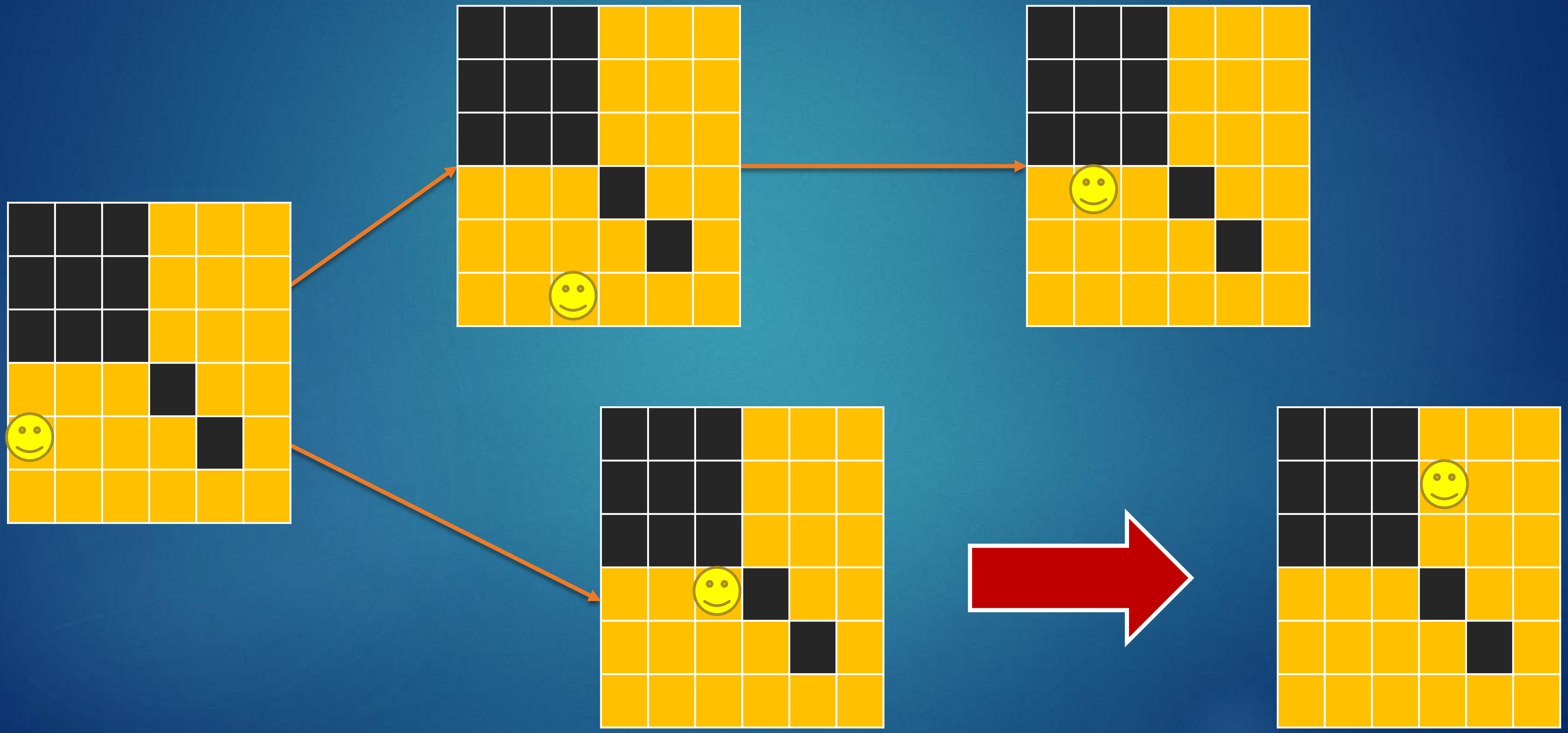


# Piece Movement



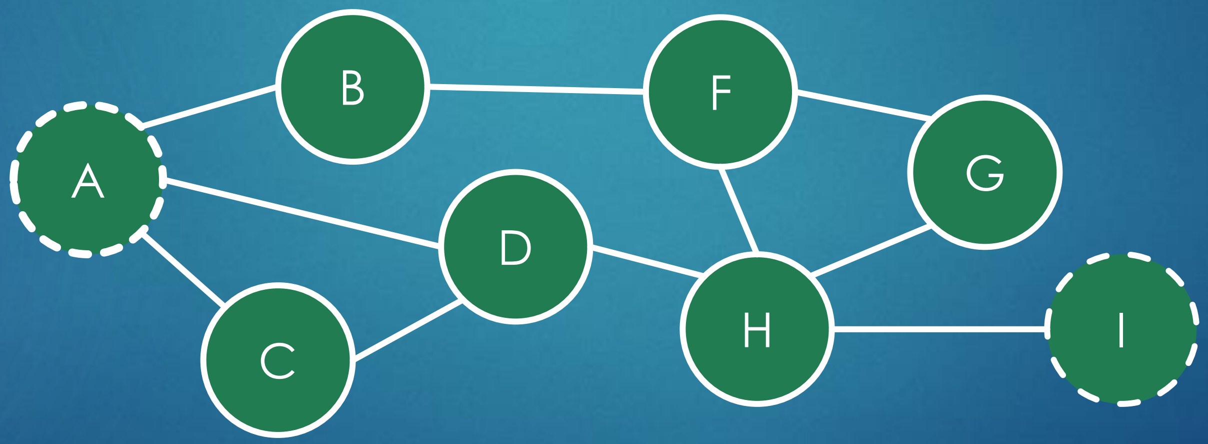


# Print States



# Intelligent Search Algorithms

## PROPOSED ARCHITECTURE



# Proposed Architecture

**Structure**

Structure (Node – State)
{

//define data structure attributes of the game:
**Array**

//define actions of the structure:
**Check Moves() – Get Next Sates() – Move() – Print State() – Equal() – Is Final()**

}

**Main**

Main (Game)
{

**S = new Structure()**
**L = new Logic()**

**L.UserPlay(S)**
**L.DFS(S)**
**L.BFS(S)**
**L.UCS(S)**
**L.Astar(S)**

}

**Logic**

Logic (Play Commands)
{

//define search strategies:

**UserPlay()**
**DFS()**
**BFS()**
**UCS()**
**Astar()**

}

# Proposed Architecture

Structure

//define actions of the structure:

**Check Moves(): Get all possible moves of the piece.**

**Move(): Apply a move at a specific position.**

**Get Next Sates(): generate N structure(state - node) objects <u>by copying current structure</u> (by values – deep copy) – where N is the number of possible moves from Check Moves() , then apply move() for all generated objects using new positions from Check Moves().**

**Print State(): print the structure attributes values.**

**Equal(): check the equality of tow states(nodes - Structure) by values(deep check)**

**Is Final(): check if the current state(node - structure) represents the goal of the game**

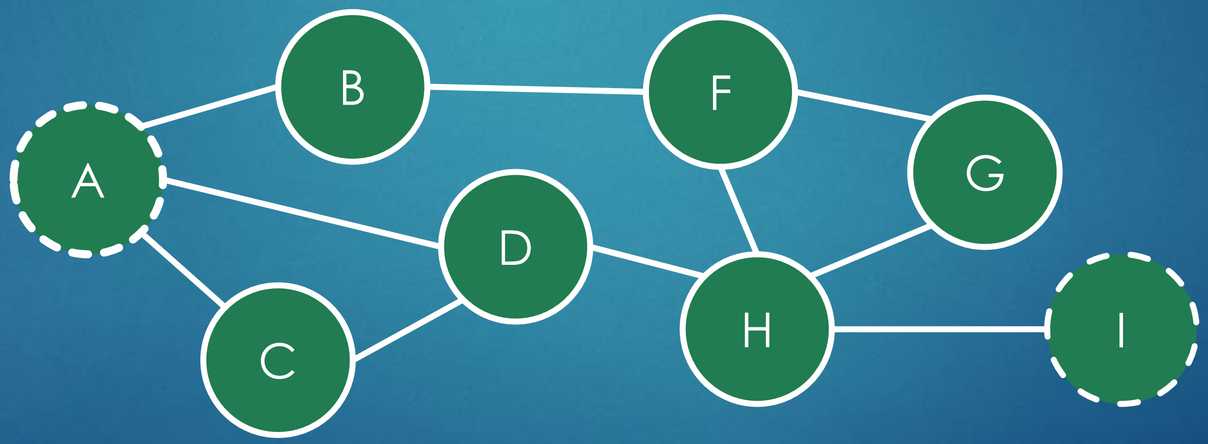
# Proposed Architecture

Structure

```
List<Position> Check Moves(){ //check up – down – left – right positions}

Move(Position){ //change the position of the piece}

List <Structure> Get Next Sates(){
      List <Structure> Next_States = new List of Structure;
      Possible_Positions = Check Moves();
      For each possible_position in Possible_Positions {
            Structure S = Deep Copy();
            S.Move(possible_position);
            Next_States.Add(S);
      }
      return Next_States;
}

Print State(){ //print the structure attributes values}

Equal(Structure S){ //check the equality of tow states(nodes - Structure) by values(deep check)}

Is Final(){ //check if the current state(node - structure) represents the goal of the game}

Structure Deep Copy(){ //copy all attributes values of current structure to the new generated state(node - structure)}
```

# Intelligent Search Algorithms

## IMPLEMENTATION – EXAMPLES



# Final Execution – Command Line App

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 8 | 8 | | | 9 | 9 |
| 1 | | 10 | 10 | | | 7 |
| 0 | 0 | 0 | 3 | | 5 | 7 |
| 2 | 11 | 11 | 3 | | 5 | 6 |
| 2 | | 4 | 4 | 12 | 12 | 6 |

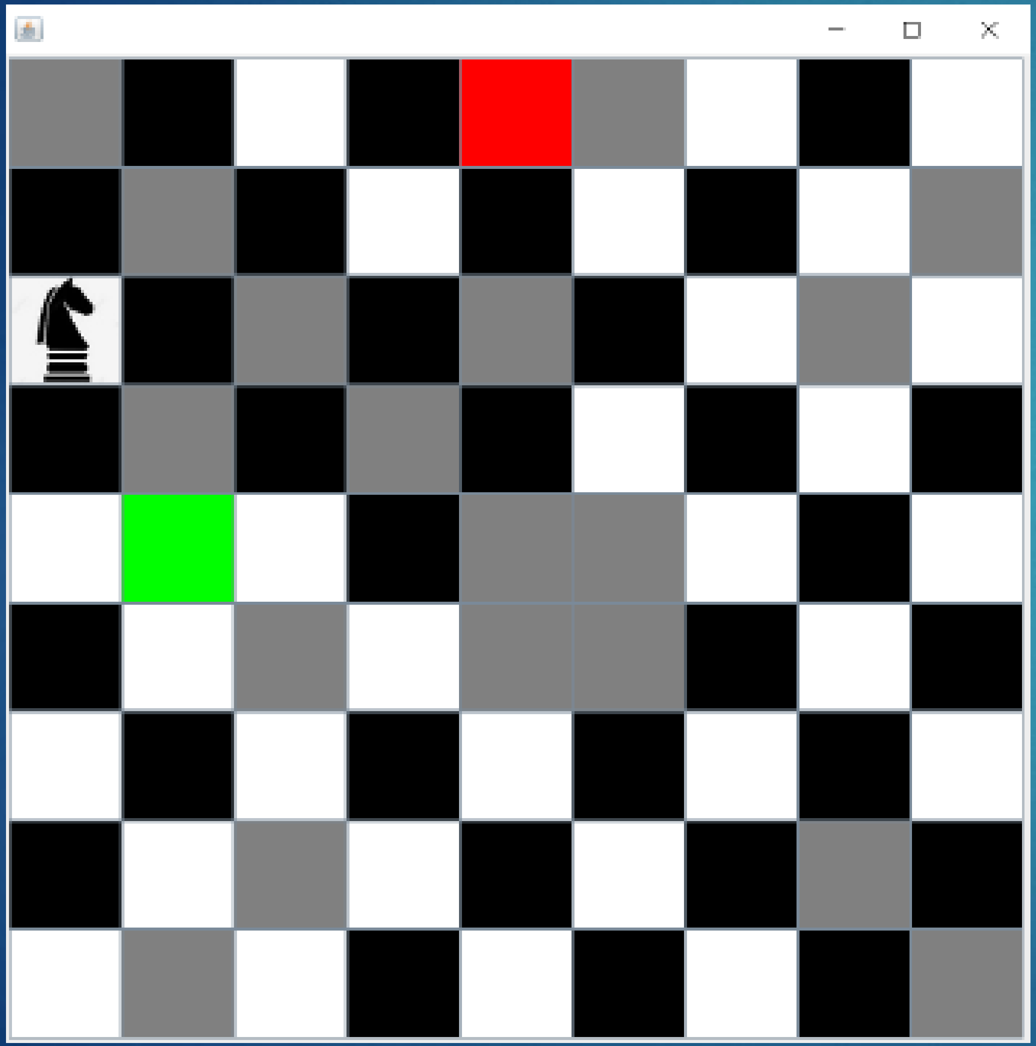
```
1) User Commands.
2) DFS.
3) BFS.
4) UCS.
5) A*.
6) Exit.
Enter strategy you want to play with: 3
searching for solution...
```

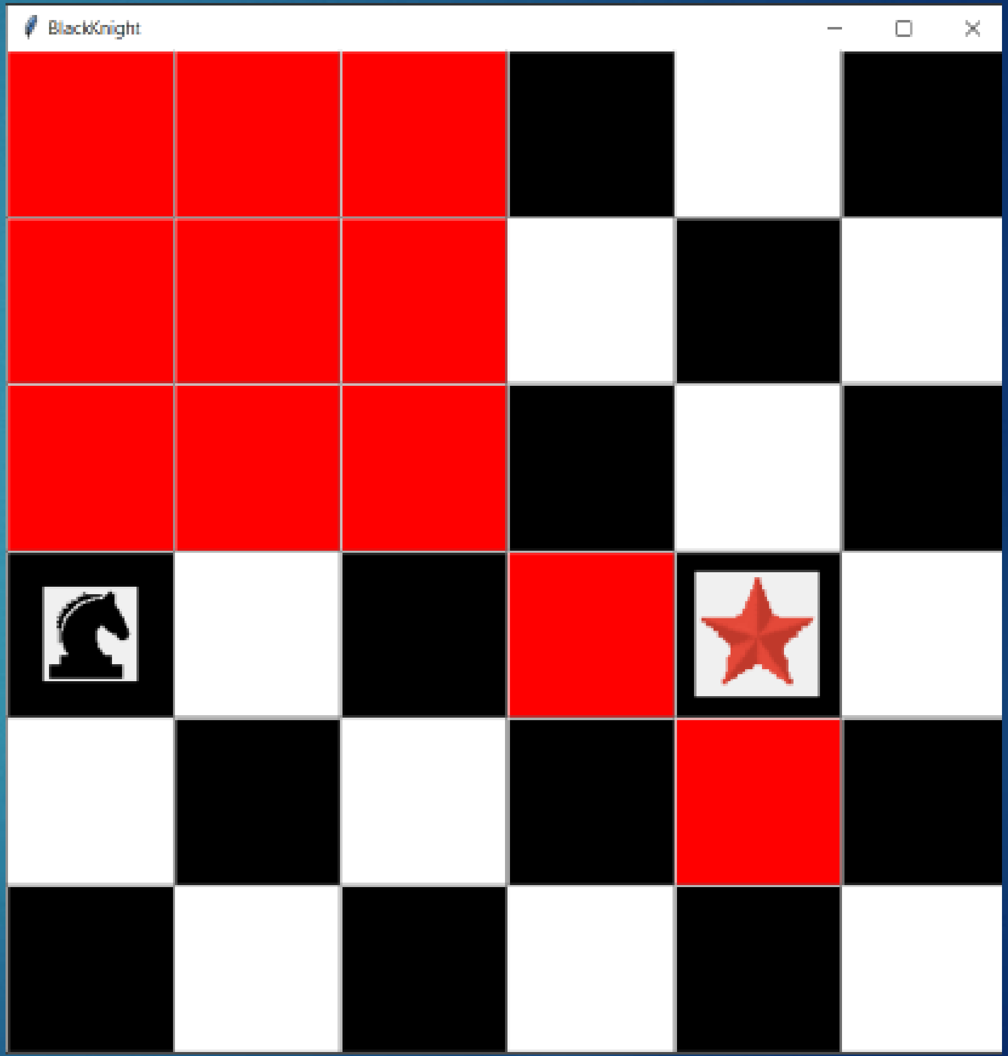
# Final Execution – Command Line App

```
----------------------------- created level 2 id done ---------------------------------
steps: 0
----------------------------
| 2  |    |    | 8  | 8  | 12 | 12 |
| 2  |    | 9  | 9  |    |    | 6  |
| 1  | 1  | 1  | 4  | 5  |    | 6  |
| 3  | 10 | 10 | 4  | 5  |    | 7  |
| 3  |    | 11 | 11 | 13 | 13 | 7  |
----------------------------
*****************************************************
***********************   A Star  ******************************
take time: 0.873 s
visited 486
steps: 16
----------------------------
| 8  | 8  | 12 | 12 | 5  |    | 6  |
| 2  |    | 9  | 9  | 5  |    | 6  |
| 2  |    |    |    | 1  | 1  | 1  |
| 3  | 10 | 10 | 4  |    |    | 7  |
| 3  | 11 | 11 | 4  | 13 | 13 | 7  |
----------------------------
***********************   A Star End ******************************
***********************   Ucs  ******************************
take time: 30.758 s
visited 2842
steps: 16
----------------------------
| 8  | 8  | 12 | 12 | 5  |    | 6  |
| 2  |    | 9  | 9  | 5  |    | 6  |
| 2  |    |    |    | 1  | 1  | 1  |
| 3  | 10 | 10 | 4  |    |    | 7  |
| 3  | 11 | 11 | 4  | 13 | 13 | 7  |
----------------------------
***********************   Ucs End ******************************
```

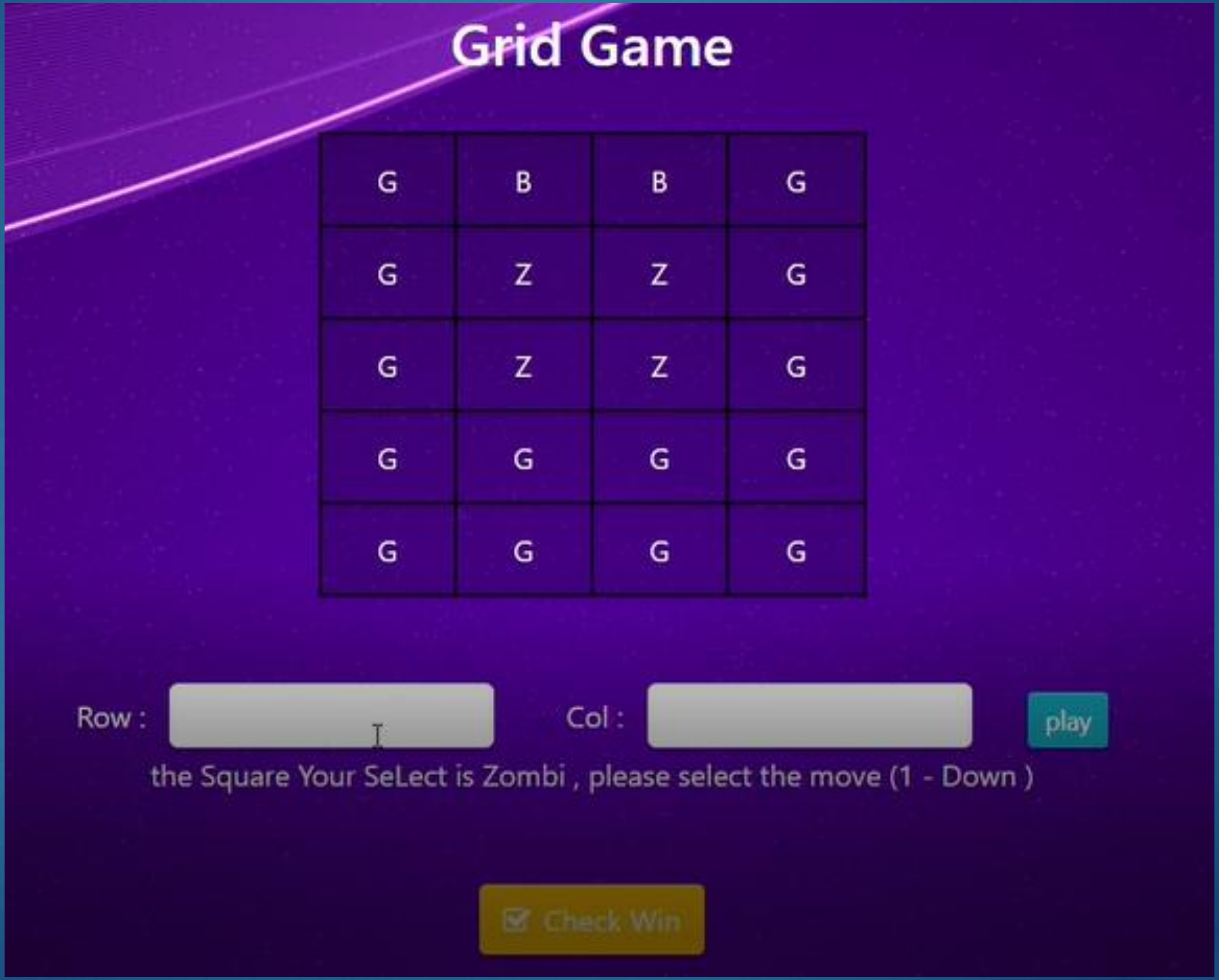
# Final Execution – Game App

# Final Execution – Desktop App





# Final Execution – Web App



# Final Execution – Mobile App

# Intelligent Search Algorithms

Thank You