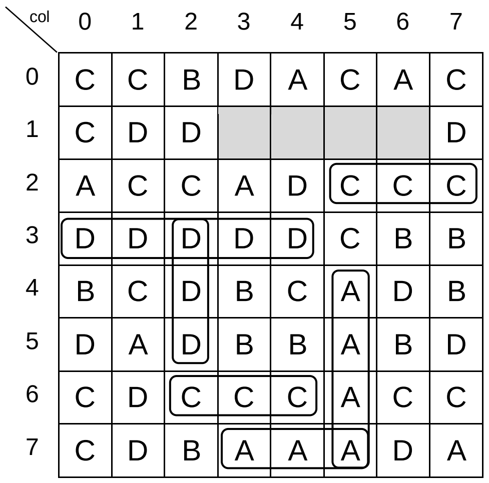
Assignment 6: Star Pickers

Due: 23:59, Sat 10 Dec 2022 Full marks: 100

Introduction

The objective of this assignment is to let you practice the usage of inheritance, polymorphism, dynamic casting, and basic file I/O. Your task is to write some classes and a client program to implement a board game that we conceive and call *Star Pickers*. We borrow some terminology from the *Star Wars* novel to dress up the game.

In the board game, an 8 × 8 grid contains treasures known as "*stars*" in the squares. The stars are represented by uppercase letters A–Z. There are two teams of players to compete to grab as many stars as possible from the board. The team getting more stars will win the game. A player may grab a whole horizontal or vertical sequence of two or more stars of the same letter, e.g., AAA, BBB, etc. in one turn. An example board and its ASCII-based representation are shown below in Figure 1. We highlighted some good picks (horizontal and vertical sequences of three or more stars of the same symbol) enclosed in round rectangles in the figure. In the program, the board is represented by a 2-D `char`-array. After the stars (letters) are taken from the board, their containing cells become empty cells, which are represented by the dot characters in the array. For `x` and `y` in the range [0, 7], each cell at position (y, x) on the board corresponds to the array element `cells[y][x]`. This array is a data member of a `Board` object (its details to come in next section).



Console Representation
(Dots represent empty cells.)

```
  0 1 2 3 4 5 6 7
0 C C B D A C A C
1 C D D . . . . D
2 A C C A D C C C
3 D D D D D C B B
4 B C D B C A D B
5 D A D B B A B D
6 C D C C C A C C
7 C D A A A A D A
```

Figure 1: An example game board with good picks enclosed in round rectangles, and its console representation

Each player is called a *star picker*, who have different skill levels. Table 1 shows the terms we use to denote each type of star pickers and summarize their skills.

Table 1: Different subtypes of star pickers

Type	**Picking Skill Level**	**Directions of Picking**	**Default `maxStars`***	**Skills**
Youngling	Low	Horizontal only	3	Due to their young age, younglings are not able to pick too many stars. They can pick horizontal lines of up to `maxStars` same stars only.
Padawan	High	Horizontal or vertical	6	Padawan are more well-trained than younglings and can pick more stars at a time. They can pick horizontal or vertical lines of up to `maxStars` same stars.
Master	Best	Horizontal or vertical	Board size	Masters are the most skillful type of star pickers. The `maxStars` limit does not apply to them. They can pick the entire row or column of stars in the grid at a time. Apart from picking stars, they also have a special ability to select two rows in the grid to swap before picking any stars. This skill can help improve the reward of his pick or his teammates' picks, or can be used to spoil good picking targets of the opponent's round to come next.

* `maxStars` refers to the maximum number of stars (of the same letter) that can be picked by the star picker at one pick. It is a data member in the Youngling class.

We will create three classes to model these three types of players. Using inheritance, we can achieve code reuse to save coding effort in the subclasses. We first define an abstract base class called `StarPicker` as the top-level interface in the class hierarchy. The following class diagram shows the relationship between the classes.

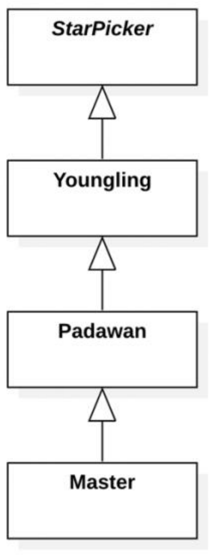


Figure 2: The class diagram for this program

Program Specification

You are to write a program which consists of the source files listed in Table 2 below.

Table 2: Header and source files of this program

	File	Description	Remarks
1	*StarPicker.h*	The *StarPicker* class interface (an abstract base class)	Provided. Don't change the content of these files.
2	Youngling.h	The Youngling class interface (inheriting StarPicker)	
3	Padawan.h	The Padawan class interface (inheriting Youngling)	
4	Master.h	The Master class interface (inheriting Padawan)	
5	Board.h	The Board class interface	
6	*StarPicker.cpp*	An abstract base class implementation	
7	Youngling.cpp	The *StarPicker* class implementation	You are to create these source files by yourself.
8	Padawan.cpp	The Youngling class implementation	
9	Master.cpp	The Padawan class implementation	
10	Board.cpp	The Master class implementation	You are to fill in some TODO parts.
11	game.cpp	The client program (containing the main function)	

All the header files and implementation of StarPicker.cpp have been fully given to you.

Class Board (Board.h, Board.cpp)

The Board class models the game board, i.e., the grid of cells containing the stars. Its interface is given as follows.

```
class Board
{
public:
    static const int W = 8;  // board width
    static const int H = 8;  // board height
    Board(const char* filename);
    void print() const;
    int wipeLine(int y, int x, int cap, bool vertical = false);
    void swapRows(int r1, int r2);
    int getStars() const;
private:
    int stars;
    char cells[H][W];
    void loadFromFile(const char* filename);
};
```

Data Members

int stars;

The count of stars that are still on the board.

char cells[H][W];

The 2-D array of char type holding the letters representing the stars. W is the board's width and H is board's height. Both are static members defined in this class. In this program, they are both set to 8. But make sure your program is scalable to changes in the values of W and H.

Constructor and Member Functions

`Board(const char* filename);`

The constructor of this class performs initialization of the `cells` array. It accepts a c-string argument passed by the client to specify the file path of an input text file which defines the board's content. The content of the file will be loaded into the `cells` array if the file can be opened successfully. If the `filename` c-string passed from the client is empty (containing '\0' only), this function will skip file loading and "hardcode" the array as the following grid of letters. This arrangement is to ease our testing effort such that we don't need an input file for every round of program testing.

```
  0 1 2 3 4 5 6 7
0 A B B B B B B B
1 A C C C C D D D
2 A E E E B E C B
3 A B E B B E C B
4 A B E D D E C B
5 A C E F F E F B
6 A D A B B E C B
7 A C A B B D C B
```

Figure 3: The hardcoded board content in the Board constructor

`void print() const;`

This function is to print the game board.

`int wipeLine(int y, int x, int cap, bool vertical = false);`

This function is to facilitate a pick action (support the implementation of the pick() virtual method defined in the various subclasses of `StarPicker`). The return value is the score gained by the pick action, which equals the number of stars grabbed by this pick which starts at position (y, x).

First, it checks whether (y, x) may fall off the board (e.g., negative indexes or indexes greater than H and W). If so, it returns 0 as the score obtained. Another case is that if (y, x) is targeting an empty cell (a dot symbol instead of A-Z), then it also returns 0. This is the validation part. This time, no error messages are required for off-the-board or empty cell accesses.

Then it *wipes* the cells of the same letter starting at (y, x) up to the cell at distance of `cap` apart. The Boolean parameter `vertical` is used to control whether it is to process the cells in the horizontal line, from left to right, or the cells in the vertical line, from top to bottom, which are next to (y, x). For example, suppose (y, x) = (2, 5), `cap` = 4, and `vertical` = true, then this pick will grab the 4 letter-E stars as highlighted in yellow and replaces the letters with dots (marking them empty).

```
  0 1 2 3 4 5 6 7
0 A B B B B B B B
1 A C C C C D D D
2 A E E E B E C B
3 A B E B B E C B
4 A B E D D E C B
5 A C E F F E F B
6 A D A B B E C B
7 A C A B B D C B
```

pick 4 stars vertically at (2, 5)

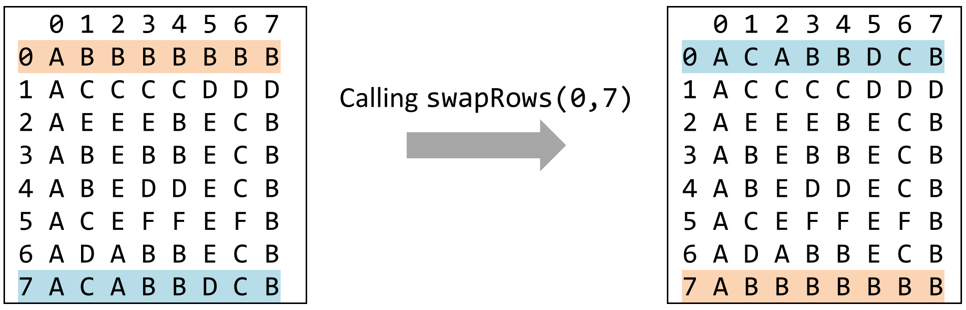
```
  0 1 2 3 4 5 6 7
0 A B B B B B B B
1 A C C C C D D D
2 A E E E B . C B
3 A B E B B . C B
4 A B E D D . C B
5 A C E F F . F B
6 A D A B B E C B
7 A C A B B D C B
```

Figure 4: An example pick realized by the wipeLine() method

In this example, the function will also deduct 4 from the `stars` data member, which keeps track of the number of stars remained on the board. The return value of the function is 4 because 4 stars are picked successfully. The caller of this function may use the return value to accumulate the total score obtained by a team thus far.

`void swapRows(int r1, int r2);`

This function is to facilitate the "swap two rows" action (which a `Master` object can perform). It swaps the two rows specified via indexes `r1` and `r2` passed from the caller. For example, Figure 5 shows the updated board content after calling `swapRows(0,7)` on the board on the left.



	0	1	2	3	4	5	6	7
0	A	B	B	B	B	B	B	B
1	A	C	C	C	C	D	D	D
2	A	E	E	E	B	E	C	B
3	A	B	E	B	B	E	C	B
4	A	B	E	D	D	E	C	B
5	A	C	E	F	F	E	F	B
6	A	D	A	B	B	E	C	B
7	A	C	A	B	B	D	C	B

Calling `swapRows(0,7)`

	0	1	2	3	4	5	6	7
0	A	C	A	B	B	D	C	B
1	A	C	C	C	C	D	D	D
2	A	E	E	E	B	E	C	B
3	A	B	E	B	B	E	C	B
4	A	B	E	D	D	E	C	B
5	A	C	E	F	F	E	F	B
6	A	D	A	B	B	E	C	B
7	A	B	B	B	B	B	B	B

Figure 5: An example row-swap realized by the swapRows() method

`int getStars() const;`

Returns the `stars` data member.

Class StarPicker (StarPicker.h, StarPicker.cpp)

The `StarPicker` class models a star picker. Its interface is given as follows.

```
class StarPicker {
public:
    StarPicker();
    string getTeam();
    void setTeam(string t);
    bool getInt(int &z);
    virtual int pick(Board &board) = 0;
    virtual string info();
private:
    string team;
};
```

Data Members

`string team;`

The data member `team` is the name of the team this star picker belongs to.

Constructor and Member Functions

`StarPicker();`

Nothing to do or initialize.

`string getTeam(); void setTeam(string t);`

The getter and setter methods for the `team` attribute. The setter assigns `team` with the argument `t`.

`bool getInt(int &z);`

This function is used to get an integer into the reference parameter `z` from the user via console input with basic validation. If the user input is an integer, this function returns true. The function caller can obtain the user input via the output parameter `z`, which is an alias of some integer variable declared on the caller side. Otherwise, the function resets the state of `cin`, discards the values that we don't want on the `cin` stream, prints an error message "Invalid input: …", and return false to the caller. You can make use of this function to compose other functions in the subclasses to keep prompting the user to enter multiple integer values for some inputs like the target cell position (y, x) on the board to access the target star or star sequence.

`virtual int pick(Board &board) = 0;`

This is a pure virtual member function to be overridden by the subclasses.

`virtual string info();`

This is a virtual member function that returns a string describing the star picker. At the `StarPicker` class level, the only information to return is the team name this player belongs to.

Class Youngling (`Youngling.h`, `Youngling.cpp`)

The `Youngling` class models younglings and is a subclass of `StarPicker`. Its interface is given as follows.

```
class Youngling : public StarPicker {
public:
    Youngling();
    Youngling(int ms);
    int getMaxStars();
    void setMaxStars(int ms);
    void getInputs(string prompt, int &y, int &x);
    virtual int pick(Board &board);
    virtual string info();
private:
    int maxStars;
};
```

Data Members

`int maxStars;`

This data member defines the maximum number of stars (of the same letter) that can be picked by the youngling in one pick.

Constructors and Member Functions

`Youngling();`

Initialize `maxStars` to 3.

`Youngling(int ms);`

Initialize `maxStars` to `ms` passed from the client code. However, in case `ms` is smaller than 1, set `maxStars` to 1, and if `ms` is greater than 3, then cap `maxStars` to 3 only.

`int getMaxStars(); void setMaxStars(int ms);`

The getter and setter methods for the `maxStars` attribute. The setter assigns `maxStars` with the argument `ms`. For simplicity, we don't implement any validation rules in the setter method for this attribute.

`void getInputs(string prompt, int &y, int &x);`

This function prints the `prompt` message (passed by the function caller) to the console to signal the user to input two integer values for `y` and `x` respectively. It keeps repeated prompting until two integers are successfully received via `cin`. The function caller can obtain the user inputs via the output parameters `y` and `x`, which are aliases of two integer variables declared on the caller side.

Hint: you can make use of the `getInt()` function defined in the StarPicker base class.

`virtual int pick(Board &board);`

This member function overrides the abstract pick() declared in the base class. It calls the getInputs() function with prompt message `"Enter (y, x): "` to get y and x from the user via console input. (y, x) is the coordinate to address the target cell on the board. Then it calls the `wipeLine()` method of the board object to grab the stars on the board. Think about how to call `wipeLine()`.

Hint: Remember that younglings can only pick stars up to `maxStars` and in horizontal lines only.

`virtual string info();`

At this subclass level, besides the team's name, include the word "Youngling" and the `maxStars` value in the information string to return. Look at sample output to see the expected format.

Class Padawan (`Padawan.h`, `Padawan.cpp`)

The `Padawan` class models padawans and is a subclass of `Youngling` (for reusing its attributes or methods). Its interface is given as follows.

```
class Padawan : public Youngling {
public:
    Padawan();
    Padawan(int ms);
    void getInputs(string prompt, int &y, int &x, char &dir);
    virtual int pick(Board &board);
    virtual string info();
};
```

Data Members

None.

Constructors and Member Functions

`Padawan();`

Initialize `maxStars` to 6. (How?)

`Padawan(int ms);`

Initialize `maxStars` to `ms` passed from the client code. However, in case `ms` is smaller than 1, set `maxStars` to 1, and if `ms` is greater than 6, then cap `maxStars` to 6 only.

`void getInputs(string prompt, int &y, int &x, char &dir);`

This function prints the `prompt` message (passed by the function caller) to the console to signal the user to input two integer values for `y` and `x` respectively, and one character for `dir`. For `dir` (i.e., direction), its expected value must be either 'h' or 'v' (lowercase) which represents "horizontal" and "vertical" respectively. If other values are received, its prints an error `"Invalid input for direction!"`. The function keeps repeated prompting until all two integers and either 'h' or 'v' are successfully received via `cin`. The function caller can obtain the user inputs via the output parameters `y`, `x`, `dir` which are aliases of two integer variables and one character variable declared on the caller side.

Hint: is it possible to reuse the `getInputs()` function in the superclass of the class?

`virtual int pick(Board &board);`

This member function overrides the superclass's pick(). It calls the getInputs() function with prompt message `"Enter (y, x, h/v): "` to get y, x and `dir` from the user via console input, and uses them to make a call to the `wipeLine()` method of the board object to grab the stars on the board. Think about how to call `wipeLine()`.

Hint: Remember that padawans can pick stars up to `maxStars` in either direction.

`virtual string info();`

At this subclass level, besides the team's name, include the word "Padawan" and the `maxStars` value in the information string to return. Look at sample output to see the expected format.

Class Master (`Master.h`, `Master.cpp`)

The `Master` class models masters and is a subclass of `Padawan` (for reusing its attributes or methods). Its interface is given as follows.

```
class Master : public Padawan {
public:
    Master();
    void getInputs(string prompt, char &swap, int &r1, int &r2);
    virtual int pick(Board &board);
    void magicSwap(Board &board);
    virtual string info();
};
```

Data Members

None.

Constructor and Member Functions

`Master();`

Initialize `maxStars` to the maximum of Board's H and W. (How?) This is to make a master be able to grab stars of the entire row or column. This are no parametrized constructors for the client to customize the `maxStars` value because this limit does not apply to a master.

`void getInputs(string prompt, char &swap, int &r1, int &r2);`

This function prints the `prompt` message (passed by the function caller) to the console to signal the user to input one character for `swap` first, followed by two integer values for `r1` and `r2` respectively, and. For `swap`, its expected value must be either 'y' or 'n' (lowercase) which represents "yes" or "no" respectively. The function keeps repeated prompting until either 'y' or 'n' and two integers are successfully received via `cin`. The function caller can obtain the user inputs via the output parameters `swap`, `y`, `x`, which are aliases of one character variable and two integer variables declared on the caller side.

`virtual int pick(Board &board);`

This implements the Master's version of the pick action, which is now basically no different from its superclass Padawan's pick(). (Then why keep this function here? From an OO design viewpoint, it is better to define it here for easier extensibility in the future, say, if a master knows how to pick stars in diagonal lines over padawans one day, then we just add more code to this function body.) How can you reuse the Padawan's pick() to greatly simplify the implementation here.

`void Master::magicSwap(Board &board);`

It prompts the user with message `"Swap rows? "` using the `getInputs()` function defined above. If the `swap` answer is 'y', then it calls the board's swapRows() method passing the user-entered row indexes `r1` and `r2`.

`virtual string info();`

At this subclass level, besides the team's name, include the word "Master" but skip the `maxStars` value in the information string to return. Look at sample output to see the expected format.

The Client Program (game.cpp)

The client program first asks the user for the file name of an input text file for loading the board content. The user may simply press the Enter key to skip file loading and use the hardcoded initial board. Then it prompts the user for the number of rounds to play.

In the client program, there are two teams of three star-pickers. One is labeled "Jedi", and the other labeled "Sith". Each team is implemented as a vector of `StarPicker` pointers pointing to one Youngling object, one Padawan object and one Master object. The two teams will run the game in rounds. In one round, every player of the same team grabs stars from the board by their skills. So, one round equals a series of three picking actions done by a team. When the current round finishes, the turn is passed to the opponent team to run their round (3 picking actions). This goes on until the board has no more stars or the number of rounds entered via console input at the program start is reached.

All client code except one function's body has been provided to you. You need to implement the following function:

`int runRound(Board &board, vector<StarPicker*> &team);`

This function loops over the input vector `team`, and print the information string of each team member. It checks if the current vector element is pointing to a `Master` object. If so, it calls the `magicSwap()` function, and print the board to show the board content after rows swap. Of course, the most important action to do here is to call the `pick()` function of each subclass via the vector element of `StarPicker*` type. After each pick action, the board should be printed. The function also accumulates the scores obtained by each team member's pick and return the total back to the caller.

File input

Some sample input files for loading the initial board content can be found on Blackboard. Below is one example:

```
A A A B B E C B
A B C D D E C E
B B D D D E E C
A B D C A D B C
A B C D D E E E
D A B A B B A D
A A B D C C B A
C B B B A A A C
```

The file format is having one space between each pair of letters. The number of letters per line is always equal to Board::W and the number of file records (rows) in the file is always equal to Board::H. Now, both H and W are 8 but your code should be written scalable to changes to H and W.

`void Board::loadFromFile(const char* filename)`

Use `ifstream` object to open the file specified by filename. Print "File could not be opened" if there exists failure when opening the file and call exit(1) to stop the program immediately. Perform file reading operations to load each letter into the `cells` array of Board.

Reminders

- For more details of how to get started, read the starter code provided on Blackboard.
- When you write the classes, implement the member functions and test them individually one by one. Your classes could be graded separately, so you should not mix the functionalities between classes or do something other than specified under each function.
- Don't use any global variables.

Sample Runs

In the following sample runs, the blue text is user input and the other text is the program printout. You can try the provided sample program for other input. *Your program output should be exactly the same as the sample program* (same text, symbols, letter case, spacings, etc.). Note that *there is a space after the ':' in the program printout*.

```
Enter board file name: ↵
How many rounds to play? 100↵
Initial board:
  0 1 2 3 4 5 6 7
0 A B B B B B B B
1 A C C C C D D D
2 A E E E B E C B
3 A B E B B E C B
4 A B E D D E C B
5 A C E F F E F B
6 A D A B B E C B
7 A C A B B D C B
Round 1:
Sith: Youngling, maxStars: 3
Enter (y, x): 1 5↵
  0 1 2 3 4 5 6 7
0 A B B B B B B B
1 A C C C C . . .
2 A E E E B E C B
3 A B E B B E C B
4 A B E D D E C B
5 A C E F F E F B
6 A D A B B E C B
7 A C A B B D C B
Stars left: 61
Sith: Padawan, maxStars: 6
Enter (y, x, h/v): 0 1 h↵
  0 1 2 3 4 5 6 7
0 A . . . . . . B
1 A C C C C . . .
2 A E E E B E C B
3 A B E B B E C B
4 A B E D D E C B
5 A C E F F E F B
6 A D A B B E C B
7 A C A B B D C B
Stars left: 55
Sith: Master
Swap rows? n↵
  0 1 2 3 4 5 6 7
0 A . . . . . . B
1 A C C C C . . .
2 A E E E B E C B
3 A B E B B E C B
4 A B E D D E C B
5 A C E F F E F B
6 A D A B B E C B
7 A C A B B D C B
Enter (y, x, h/v): 0 0 v↵
  0 1 2 3 4 5 6 7
0 . . . . . . . B
```

```
1 . C C C C . . .
2 . E E E B E C B
3 . B E B B E C B
4 . B E D D E C B
5 . C E F F E F B
6 . D A B B E C B
7 . C A B B D C B
Stars left: 47
Score: Jedi 0 vs. Sith 17
Round 2:
Jedi: Youngling, maxStars: 3
Enter (y, x): 2 1↵
  0 1 2 3 4 5 6 7
0 . . . . . . . B
1 . C C C C . . .
2 . . . . B E C B
3 . B E B B E C B
4 . B E D D E C B
5 . C E F F E F B
6 . D A B B E C B
7 . C A B B D C B
Stars left: 44
Jedi: Padawan, maxStars: 6
Enter (y, x, h/v): 2 5 v↵
  0 1 2 3 4 5 6 7
0 . . . . . . . B
1 . C C C C . . .
2 . . . . B . C B
3 . B E B B . C B
4 . B E D D . C B
5 . C E F F . F B
6 . D A B B . C B
7 . C A B B D C B
Stars left: 39
Jedi: Master
Swap rows? y 0 1↵
  0 1 2 3 4 5 6 7
0 . C C C C . . .
1 . . . . . . . B
2 . . . . B . C B
3 . B E B B . C B
4 . B E D D . C B
5 . C E F F . F B
6 . D A B B . C B
7 . C A B B D C B
Enter (y, x, h/v): 1 7 v↵
  0 1 2 3 4 5 6 7
0 . C C C C . . .
1 . . . . . . . .
2 . . . . B . C .
3 . B E B B . C .
```

```
4 . B E D D . C .
5 . C E F F . F .
6 . D A B B . C .
7 . C A B B D C .
Stars left: 32
Score: Jedi 15 vs. Sith 17
```

…
(some output skipped)
…

```
Round 8:
Jedi: Youngling, maxStars: 3
Enter (y, x): 5 6↵
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . . . . . .
3 . . . . . . . .
4 . . . . . . . .
5 . . . . . . . .
6 . . . . . . C .
7 . . . . . D . .
Stars left: 2
Jedi: Padawan, maxStars: 6
Enter (y, x, h/v): 6 6 h↵
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . . . . . .
3 . . . . . . . .
4 . . . . . . . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . D . .
Stars left: 1
Jedi: Master
Swap rows? n↵
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . . . . . .
3 . . . . . . . .
4 . . . . . . . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . D . .
Enter (y, x, h/v): 7 5 h↵
```

```
  0 1 2 3 4 5 6 7
0 . . . . . . . .
1 . . . . . . . .
2 . . . . . . . .
3 . . . . . . . .
4 . . . . . . . .
5 . . . . . . . .
6 . . . . . . . .
7 . . . . . . . .
Stars left: 0
Score: Jedi 29 vs. Sith 35
Game Over!
Sith wins!
```

Use our provided sample executables to provide more outputs for your testing and comparison.

Submission and Marking

- Your program file names should be `Youngling.cpp`, `Padawan.cpp`, `Master.cpp`, `Board.cpp`, and `game.cpp`. Submit the *five files* in Blackboard (https://blackboard.cuhk.edu.hk/). We assume that you won't modify the .h files (which are not recommended and may lead to mark deductions), but in case you really need to modify them, please submit all your modified .h files.
- You can surely create other client program versions to test the game with more star pickers with different initialized team name or `maxStars`. But when you submit the game.cpp file, make sure that the other code besides the `runRound()` function in that file stays the same as what we provided in the starter code version for our easier grading although we may also plug in other client versions on our own design to test your classes.
- Insert *your name*, *student ID*, and *e-mail* as comments at the beginning of all your files.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be *free of compilation errors and warnings*.
- Your program should *include suitable comments as documentation*.
- ***Do NOT plagiarize.*** Sending your work to others is subject to the same penalty for copying work.