UCLA

CS 31 : Introduction To Computer Science I

Howard A. Stahl

1

Project 7

- The Goal: A Working Wordle Game
- Background: Please Play A Few Games With This Free Game
  - https://www.nytimes.com/games/wordle/index.html
- Truth In Advertising:
  - In Order To Fully Support The Game, There Are Quite Abit I Made For You…
  - 8 Tries Rather 5
  - I Curated A WordList Dictionary Of About 3,800 Words

2

Project 7

- Unlike Earlier Assignments, I Am Supplying You With A Partial "Skeleton" Of The Code Solution
- It Will Run Right Out Of The Box
  - Some Important Pieces Are Stubbed Out…
  - These Are The Parts You Need To Complete
- Hint 1: Acquire The Skeleton!
- Hint 2: Build And The Run The Skeleton!
  - Look At What Is Working And What Is Not

3

Project 7

- The Work Product: The Implementation Of The Public API Of The Classes Described Here And In The Assignment.
- You Are Free To Do It However You Like, But You Must Provide The Public API I Am Looking For...
  - You Can Add Classes, Methods, Members As You Feel Appropriate
  - But I Honestly Don't Think You'll Need To...
- In What Follows, It Is The **Bolded** Portions That You Need To Complete

4

Some Stuff Is Completely Done...

- A Piece Represents An Individual Letter Played

<enumeration> **LETTER**
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z NOTVALID

Piece
- mLetter : LETTER
+ Piece( ) + Piece(c : char) + Piece(s : string) + getLetter( ) : LETTER + getLetterAsString( ) : string

5

Some Stuff Is Completely Done...

- A Piece Represents An Individual Letter Played

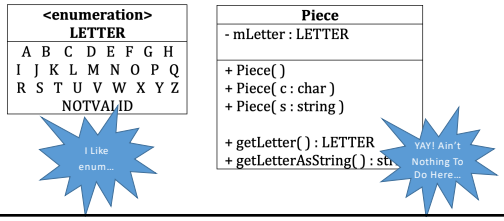
<enumeration> **LETTER**
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z NOTVALID

Piece
- mLetter : LETTER
+ Piece( ) + Piece(c : char) + Piece(s : string) + getLetter( ) : LETTER + getLetterAsString( ) : string

6

Some Stuff Is Completely Done…

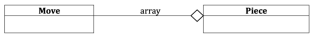
- A Piece Represents An Individual Letter Played



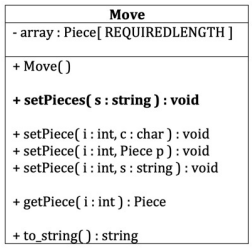
7

The Class Move

- A Move Is An Ordered Set Of Five Pieces

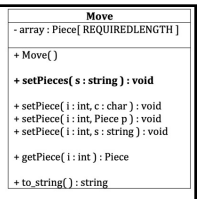


- A Move "HAS" Pieces



8

The Class Move



- A Move
  - Holds An Ordered Set Of Pieces
  - Can Be Printed By Calling to_string( )
  - `Piece getPiece( int i )`
    - An Individual Piece Can Be Acquired From A Move
    - If i Is Out Of Range, A std::logic_error Will Be Thrown…
  - `void setPiece( int i, char c )`
    - An Individual Piece Can Be Saved Into The Move
    - If i Is Out Of Range, A std::logic_error Will Be Thrown…
  - `void setPiece( int i, string s )`
    - An Individual Piece Can Be Saved Into The Move Using Just The First Letter Of The String
    - If i Is Out Of Range, A std::logic_error Will Be Thrown…
  - `void setPiece( int i, Piece p )`
    - An Individual Piece Can Be Saved Into The Move
    - If i Is Out Of Range, A std::logic_error Will Be Thrown…

9

The Class Move

- A Move Object
  - Holds An Ordered Set Of Pieces
  - Can Be Printed By Calling to_string( )
  - **`void setPieces( string s )`**
    - Build The Move From The First Five Letters Of The String
    - If The String Has More Than Five Letters, Throw std::logic_error

Move
- array : Piece[REQUIREDLENGTH]
+ Move( ) **+ setPieces(s : string) : void** + setPiece(i : int, c : char) : void + setPiece(i : int, Piece p) : void + setPiece(i : int, s : string) : void + getPiece(i : int) : Piece + to_string( ) : string

10

The Class Move

- A Move Object
  - Holds An Ordered Set Of Pieces
  - Can Be Printed By Calling to_string( )
  - **`void setPieces( string s )`**
    - Build The Move From The First Five Letters Of The String
    - If The String Has More Than Five Letters, Throw std::logic_error

Move
- array : Piece[REQUIREDLENGTH]
+ Move( ) **+ setPieces(s : string) : void** + setPiece(i : int, c : char) : void + setPiece(i : int, Piece p) : void + setPiece(i : int, s : string) : void + getPiece(i : int) : Piece + to_string( ) : string

11

The Class Score

- A Score Object Is Provided A Move And The Winning Answer
- A Score Object Determines The Answer Value For Each Played Piece

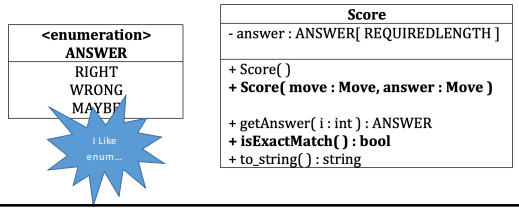
<enumeration> **ANSWER**
RIGHT WRONG MAYBE

Score
- answer : ANSWER[REQUIREDLENGTH]
+ Score( ) **+ Score(move : Move, answer : Move)** + getAnswer(i : int) : ANSWER **+ isExactMatch( ) : bool** + to_string( ) : string

12

The Class Score

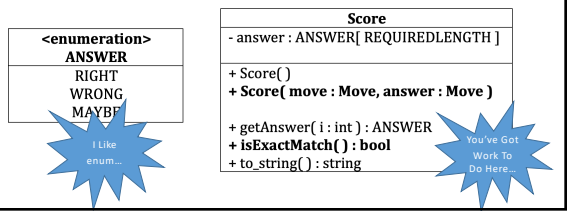
- A Score Object Is Provided A Move And The Winning Answer
- A Score Object Determines The Answer Value For Each Played Piece



13

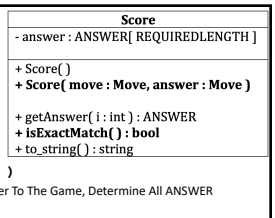
The Class Score

- A Score Object Is Provided A Move And The Winning Answer
- A Score Object Determines The Answer Value For Each Played Piece



14

The Class Score



- A Score Object
  - Holds An Ordered Set Of ANSWER Values
  - Can Be Printed By Calling to_string( )
  - **`Score( Move move, Move answer )`**
    - Provided A Played Move And The Correct Answer To The Game, Determine All ANSWER Values – RIGHT, WRONG or MAYBE
  - `Answer getAnswer( int i )`
    - An Individual Answer Can Be Acquired From A Score
    - If i Is Out Of Range, A std::logic_error Will Be Thrown...
  - **`bool isExactMatch()`**
    - Return `true` If All The ANSWERs Are The Value RIGHT; `false` Otherwise

15

The Class Score

- A Score Object
  - Holds An Ordered Set Of ANSWER Values
  - Can Be Printed By Calling to_string( )
  - **`Score( Move move, Move a`**
    - Provided A Played Move And The Correct [illegible]ne, Determine All ANSWER Values – RIGHT, WRONG or MAYBE
  - `Answer getAnswer( int i )`
    - An Individual Answer Can Be Acquired From A [illegible]core
    - If i Is Out Of Range, A std::logic_error Will Be Thrown…
  - **`bool isExactMatch()`**
    - Return `true` If All The ANSWERs Are The Value RIGHT; `false` Otherwise

Score
- answer : ANSWER[REQUIREDLENGTH]
+ Score( ) **+ Score(move : Move, answer : Move)** + getAnswer(i : int) : ANSWER **+ isExactMatch( ) : bool** + to[illegible]ring( ) : string

I Think This Is The Hardest Part…

16

The Class Score

- A Score Object
  - Holds An Ordered Set Of ANSWER Values
  - Can Be Printed By Calling to_string( )
  - **`Score( Move move, Move a`**
    - Provided A Played Move And The Correct [illegible]ne, Determine All ANSWER Values – RIGHT, WRONG or MAYBE
  - `Answer getAnswer( int i )`
    - An Individual Answer Can Be Acquired From A [illegible]core
    - If i Is Out Of Range, A std::logic_error Will Be Thrown…
  - **`bool isExactMatch()`**
    - Return `true` If All The ANSWERs Are The Value RIGHT; `false` Otherwise

Score
- answer : ANSWER[REQUIREDLENGTH]
+ Score( ) **+ Score(move : Move, answer : Move)** + getAnswer(i : int) : ANSWER **+ isExactMatch( ) : bool** + to[illegible]ring( ) : string

I Think This Is The Hardest Part…

Just Stub It Out And Leave It To The End…

17

The Class Board

- Tracks The Current Round And Manages A Set of Moves And Scores


```
          CS31Wordle Game
1 : APPLE – M_R_M
2 : AFTER – M__RM
3 : AMBER – M__RM
```

- A Board "HAS" Moves
- A Board "HAS" Scores
- These Sets Grow As Rounds Of Play Occur…

18

The Class Board

- Manages A Set of Moves And Scores
- A Board
  - Holds The Current Round Value
  - Holds An Set of Scores And Moves
  - The Game Prints The Board Via Calls To `display( )`
  - Each Call To `endRound( Move, Score )` Adds To The Set Of Moves And Scores And Increments The `mRoundCount`
  - **`Move getMoveForRound( int round )`**
    - An Individual Move Can Be Acquired From A Board
    - If round Is Out Of Range, A std::logic_error Will Be Thrown…
  - **`Score getScoreForRound( int round )`**
    - An Individual Move Can Be Acquired From A Board
    - If round Is Out Of Range, A std::logic_error Will Be Thrown…
  - **`int getCurrentRound( )`**
    - Trivial Accessor

Board
- mRoundCount : int - scores : Score[TOTALROUNDSALLOWED] - moves : Move[TOTALROUNDSALLOWED]
+ Board( ) + endRound(m : Move, s : Score) : void + display( ) : string **+ getCurrentRound( ) : int** **+ getMoveForRound(round : int) : Move** **+ getScoreForRound(round : int) : Score**

19

The Class Board

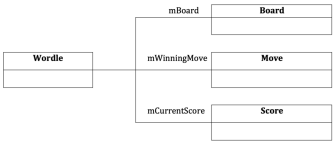
- Manages A Set of Moves And Scores
- A Board
  - Holds The Current Round Value
  - Holds An Set of Scores And Moves
  - The Game Prints The Board Via Calls To `display( )`
  - Each Call To `endRound( Move, Score )` Adds To The Set Of Moves And Scores And Increments The `mRoundCount`
  - **`Move getMoveForRound( int round )`**
    - An Individual Move Can Be Acquired From A Board
    - If round Is Out Of Range, A std::logic_error Will Be Thrown…
  - **`Score getScoreForRound( int round )`**
    - An Individual Move Can Be Acquired From A Board
    - If round Is Out Of Range, A std::logic_error Will Be Thrown…
  - **`int getCurrentRound( )`**
    - Trivial Accessor

Board
- mRoundCount : int - scores : Score[TOTALROUNDSALLOWED] - moves : Move[TOTALROUNDSALLOWED]
+ Board( ) + endRound(m : Move, s : Score) : void + display( ) : string **+ getCurrentRound( ) : int** **+ getMoveForRound(round : int) : Move** **+ getScoreForRound(round : int) : Score**

20

The Class Wordle

- Implements The Interactive Game



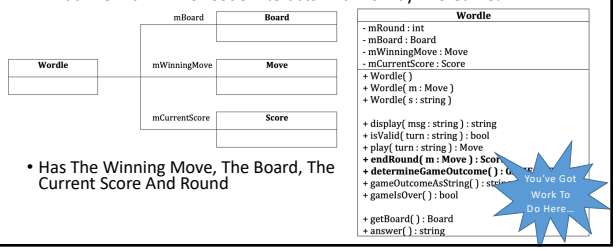
- Has The Winning Move, The Board, The Current Score And Round

Wordle
- mRound : int - mBoard : Board - mWinningMove : Move - mCurrentScore : Score
+ Wordle( ) + Wordle(m : Move) + Wordle(s : string) + display(msg : string) : string + isValid(turn : string) : bool + play(turn : string) : Move **+ endRound(m : Move) : Score** **+ determineGameOutcome( ) : GAMEOUTCOME** + gameOutcomeAsString( ) : string + gameIsOver( ) : bool + getBoard( ) : Board + answer( ) : string

21

The Class Wordle

- What The Main Driver Code Interacts With To Play The Game!



- Has The Winning Move, The Board, The Current Score And Round

22

Enumeration Used By The Wordle Class

- I Really Like Enumerations…

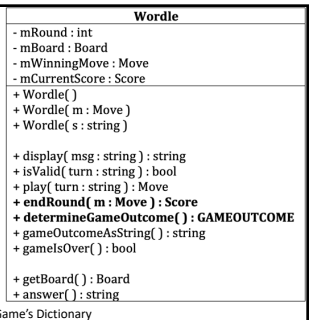
<enumeration> GAMEOUTCOME
GAMEWON
GAMELOST
GAMENOTOVER

- GAMEOUTCOME Is Used To Represent The Result Of Playing A Game

23

The Class Wordle

- Manages The Board
- Tracks The Current Round, Current Score And Winning Move
  - `bool gameIsOver( )`
    - `true` When The Game Has Ended
  - `string display( )`
    - Displays The Board As Play Progresses…
  - `Board getBoard( )`
    - Trivial Accessor For Testing Purposes…
  - `string answer( )`
    - The Winning Word…
  - `bool isValid(string turn)`
    - `true` When The String Is Found In The Game's Dictionary



24

The Class Wordle

- Manages The Board
- Tracks The Current Round, Current Score And Winning Move
  - **`GAMEOUTCOME determineGameOutcome( )`**
    - Based On The Board, Return The Appropriate GAMEOUTCOME Value
  - **`Score endRound( Move m )`**
    - Build A Score Object For This Move
    - Tell The Board The Round Has Ended
    - Update This Wordle's CurrentScore
    - Update This Wordle's Round
    - Return The Score Object You Made

Wordle
- mRound : int - mBoard : Board - mWinningMove : Move - mCurrentScore : Score
+ Wordle( ) + Wordle(m : Move) + Wordle(s : string) + display(msg : string) : string + isValid(turn : string) : bool + play(turn : string) : Move **+ endRound(m : Move) : Score** **+ determineGameOutcome( ) : GAMEOUTCOME** + gameOutcomeAsString( ) : string + gameIsOver( ) : bool + getBoard( ) : Board + answer( ) : string

25

The Class Wordle

- Why Are There Three Constructors??
  - The Version With No Arguments Randomly Picks A Word From The Game's Dictionary…
  - The Other Two Are For Cheating…

26

Random Driver Code Says:

```
• Wordle game;
  // game picks a random word…
  // off we go…
  if (game.isValid( "apple" )
  {
     Move m = game.play( "apple" );
     Score s = game.endRound( m );
     // one round is now over…
  }
  // did we win??
  if (game.gameIsOver()) { … }
```

27

Random Driver Code Says:

```
• Wordle game;
  // game picks a random word…
  // off we go…
  if (game.isValid( "apple" )
  {
     Move m = game.play( "apple" );
     Score s = game.endRound( m );
     // one round is now over…
  }
  // did we win??
  if (game.gameIsOver()) { … }
```

28

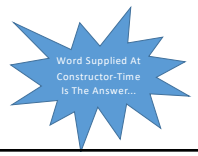
Cheating Driver Code Says:

```
• Wordle game( "apple" );
  // constructor sets the winning word…
  if (game.isValid( "apple" )
  {
     Move m = game.play( "apple" );
     Score s = game.endRound( m );
     // one round is now over…
  }
  // did we win??
  if (game.gameIsOver()) { … }
```

29

Cheating Driver Code Says:

```
• Wordle game( "apple" );
  // constructor sets the winning word…
  if (game.isValid( "apple" )
  {
     Move m = game.play( "apple" );
     Score s = game.endRound( m );
     // one round is now over…
  }
  // did we win??
  if (game.gameIsOver()) { … }
```



30

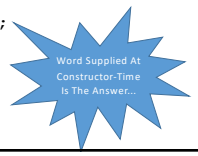
Cheating Driver Code Says:

```
• Move move;
  move.setPieces( "apple" );
  Wordle game( move );
  // constructor sets the winning word…
  if (game.isValid( "apple" )
  {
     Move m = game.play( "apple" );
     Score s = game.endRound( m );
     // one round is now over…
  }
  // did we win??
  if (game.gameIsOver()) { … }
```

31

Cheating Driver Code Says:

```
• Move move;
  move.setPieces( "apple" );
  Wordle game( move );
  // constructor sets the winning word…
  if (game.isValid( "apple" )
  {
     Move m = game.play( "apple" );
     Score s = game.endRound( m );
     // one round is now over…
  }
  // did we win??
  if (game.gameIsOver()) { … }
```



32

Putting It All Together

• main( ) Driver Code

```
Wordle game;
string turn;
do
{
     getline( cin, turn );
     if (game.isValid(turn))
     {
        Move m = game.play( turn );
        Score s = game.endRound( m );
     }
     else
        cout << turn << " was not a Dictionary word!" << endl;

     if (!game.gameIsOver())
        cout << game.display( message ) << endl;
} while( !game.isGameOver() );
```

33

Suggestions

- Read Over The FAQ
- In The Assignment, Scroll Down And Review The `assert( )` Commands…
- Start With Move...
- Then Move On To Score...
- Then Move On To Board...
- And Finish With Wordle!
- `assert( )` Each Class As You Make Progress...
- Don't Finally Play The Game Until All Your Classes Pass Their `assert( )`'s
- You Can Check Your Work To Some Degree Via CodeBoard
  - The Skeleton Has Been Loaded Into CodeBoard Already...

34