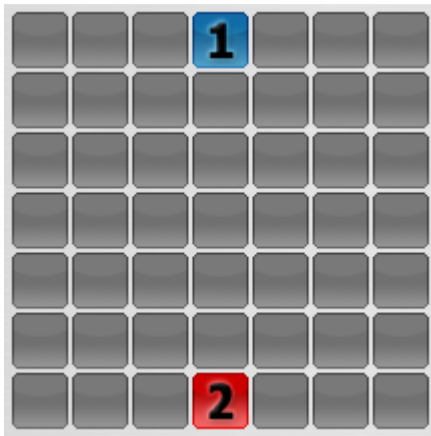
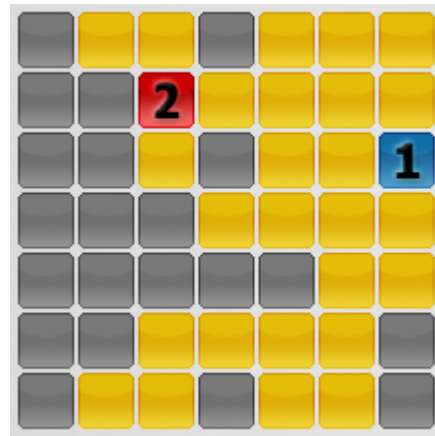


The Game

Isola is an abstract strategy board game. The board is initially filled with an arbitrary amount of squares, except for the starting position of each player. Both players have one piece; it is in the middle position of the row closest to his/her side of the board. Each turn consists of two subsequent actions: a move and a steal. A player can move his/her piece to any open neighboring square vertically, horizontally or diagonally. Next the same player will steal any other open square on the board. A stolen square cannot be occupied by either player. The first person who cannot move to any open neighboring positions loses.



A Fresh Game Board



Game Over: Player 2 Wins

Competition Rules

- The competition will follow a ladder style ranking
- Each match between two competitors will consist of 2 rounds
- Each player will get one chance to be player 1 to mitigate the small starting advantage
- The winner of a match will be the player to win both rounds
- In the event of a tie, the player with the fastest average speed across both rounds will win
- The board size will be randomly selected before each match
- The board size will always be square and odd (9x9, 15x15, 27x27, etc)
- If you attempt an invalid move or steal, an exception will be thrown.
- If any exceptions are thrown from either of your methods, your turn will be discarded and the game will continue on
- Teams of any size are permitted but all code must be submitted under one name
- The winner will have the first choice of all prizes for the entire conference (Teams may choose up to 2 prizes maximum)
- Winners must be present at the announcement of contest results to claim prizes (5:15pm, February 27th at 1505 SC)

Instructions

System Requirements

- JDK version 1.6+
- Internet connection (for downloads and submission)
- Text editor or IDE (Notepad++, Eclipse, NetBeans, etc.)

Getting Started

1. Download the `Isola.jar` file from <http://acm.uiowa.edu/uicc/>
2. Download the `SampleAI.class` file if you want to test your AI
3. Add `Isola.jar` to your system's classpath (or compile later with `java -cp`)
4. Create a new java file
5. Import `IsolaGame.IPlayer`, `IsolaGame.GameState`, and `IsolaGame.Position` into your file
6. Declare a new public class (your class name will be used to identify your AI, make it something fun and creative!) that has the same name as your `.java` file
7. Declare that your class implements the `IPlayer` interface
8. Implement the two methods `getNextMove` and `getNextSteal` from the `IPlayer` interface
9. Compile your code and use the controller to test it out!
10. If you did not add `Isola.jar` to your system's classpath, compile your code using
`javac -cp .;Isola.jar MyCoolAIName.java` (For Windows)
`javac MyCoolAIName.java -cp Isola.jar` (For Unix)
11. Submit your files when finished

Sample Java File

```
// Must NOT declare a package
import IsolaGame.IPlayer;
import IsolaGame.GameState;
import IsolaGame.Position;

// Class name must be the same as the .java file name
public class MyCoolAIName implements IPlayer {

    public Position getNextMove(GameState state){
        // My amazing code
    }

    public Position getNextSteal(GameState state){
        // My amazing code
    }

}
```

Submission

The submission deadline is **3:00 AM, February 27th 2010**. Absolutely no submissions after this time will be accepted under any circumstances. Submissions through the controller are preferred, but are also possible via e-mail if needed.

Through The Controller

1. In the UICC Competition Controller, select **File -> Submit Code...**
2. Next to the Class field, select **Choose...** and choose your class file
3. Next to the Java field, select **Choose...** and choose your java file
4. Enter the full name you used to register for the conference with in the **Name** field with a space in between (ex. "John Smith")
5. Click **Submit**
6. Verify your files were received. Look for green checkmark icons next to both of the file path textboxes. If any file uploads fail, keep trying!
7. Submitting your files more than once will overwrite any previous entries

Through E-Mail

- E-mail both your **.java** source file and your compiled **.class** file to iowa.acm@gmail.com
- Don't forget to include the name you registered for the conference with in the message subject or body
- If more than one submission is received, the most recent will be used

The API

Position

The position class is the fundamental unit of Isola. This class is used to represent a point or position on the game board in terms of a row and column number. It is similar to a coordinate (x, y) but is instead a position (row, column).

Method	Description
Position(rowNum, colNum)	Constructor for creating a new position at the specified row and column number
Position()	Constructor for creating a new position with rows and columns to be set later (defaults to 0 if not set)
int getRow()	Returns the row number the position contains
int getCol()	Returns the column number the position contains
void setRow(int row)	Updates the row number the position contains
void setCol(int column)	Updates the column number the position contains
void setRowCol(int row, int column)	Updates both the row number and the column number the position contains
boolean equals(Position position)	Compares two positions and determines if their row and column numbers are the same

IPlayer

IPlayer is the interface that each competitor must implement. Its two methods are used to obtain position objects which are used to move around the board and steal squares. All of your programming will be done within these two methods.

Abstract Methods	Description
Position getNextMove(GameState state)	Based on the game's state, this method should determine the position of your next move
Position getNextSteal(GameState state)	Based on the game's state, this method should determine the position of your next steal

GameState

GameState is the class you will use to determine what moves your AI should make. The controller will give your methods the most recent GameState object each time they are called. The state reveals the position of both players, the position of open and stolen squares, and contains a few handy methods for validation and navigating the board.

Methods	Description
char[][] getBoardInstance()	Gets the character representation of the board. The characters are contained within a 2D array. The internal representations are as follows: 1 - An open space 0 - An occupied or stolen space A - Player 1 B - Player 2
boolean isValidSteal(Position position)	Determines if a steal is valid or not. Valid steals are spaces contained within the game board that are currently open.
boolean isValidMove(Position from, Position to)	Determines if a move is valid or not. Valid moves must be to an open space that is greater than 0 but no more than 1 space away from the given position
boolean isMoveable(Position position)	Determines if a space is open (not occupied or stolen). Not to be confused with isValidMove(), as this method does not test distances or bounds.
Position Up(Position from)	Determines the position one space up from the specified position.*
Position Down(Position from)	Determines the position one space down from the specified position.*
Position Left(Position from)	Determines the position one space left of the specified position.*
Position Right(Position from)	Determines the position one space right of the specified position.*
Position UpLeft(Position from)	Determines the position one space up and left of the specified position.*
Position UpRight(Position from)	Determines the position one space up and right of the specified position.*

Position DownLeft(Position from)	Determines the position one space down and left of the specified position.*
Position DownRight(Position from)	Determines the position one space down and right of the specified position.*
int getColumns()	Gets the number of columns in the current board
int getRows()	Gets the number of rows in the current board
Position getMyPosition()	Returns your current position
Position getOpponentPosition()	Returns the current position of your opponent

** If the resulting position is not contained within the game board (out of bounds) null is returned*

*** The API has complete JavaDoc Intellisense support for IDE's*

The Controller

The controller is what will be used to run your AI against your fellow competitors. It coordinates the AI's actions, sets up and maintains the game board, and visually displays the game. It has been provided to you so you can test your AI and verify it will perform well during the competition. A sample AI has been provided to test against.

Instructions

1. To run the controller, double click the **Isola.jar** file or run it from a command prompt
2. Under the **Load AI** panel, click **Set...** for Player 1 or Player 2
3. Browse to the location of your AI's .class file
4. Select the file and click **Open**
5. Repeat steps 2-4 with the sample AI, except for the opposite player
6. Set the desired board size by clicking **Randomize**, or by scrolling to the desired size (must be an odd number between 7 and 27 inclusive)
7. Adjust the game speed to the desired setting
8. Click **Auto Run** to run the game until someone wins
9. Alternatively, you can use **Step** to perform one action at a time

Notes

- You can click on any square on the game board to display its position
- Auto Run can be paused (and resumed), allowing you to Step actions one by one in between runs if needed
- The sample AI can play against itself! Try loading the sample AI into both player slots to see how it performs
- All actions are recorded in the action window so you can see a history of all moves and steals made during the current round
- Every time your methods are called from the controller, the total time it takes for your code to execute is recorded at nanosecond precision. This AI "speed" is used to resolve ties and encourage code efficiency and optimization
- If any errors are displayed when setting an AI or setting an AI does nothing when a file is selected, ensure the class you're trying to load is part of the default package. To make sure your class is a member of the default package, remove any **package** declarations in your java file, recompile and try again
- You should reload your class file if you recompile to ensure the controller is using your most recent file
- After the conference, visit <http://acm.uiowa.edu/uicc/> for a video of the competition to see how your AI performed!