Assignment 2

Sample Answers and Marking Guide

CompSci 230 S2 2015

Clark Thomborson

Version 1.1 of 7 May 2015.

Total marks on this assignment: 50. This assignment counts 5% of total marks in COMPSCI 230.

Part 1: Code Inspection and Reverse Engineering

1) ***(5 marks)*** *Assessed work:*

   a) *Inspect the source code of hbbv2.0 to discover how its classes are related. Modify the class diagram of Figure 1 (on the next page), so that your modified diagram accurately shows all inheritances, realisations (a.k.a. implementations), and associations between classes. Associations must show navigability and multiplicity. Your diagram must show at least one specialised association (i.e. an aggregation or composition), and you will document your decision to specialise this association in part b of this question. Your modified diagram should not show any additional instance variables or methods.*

   Submit *your modified class diagram as a figure in your submission document. This figure should have a caption indicating that it is your answer to question 1a, and that it depicts "the relationships between classes in version 2.0 of the Huckle Buckle codebase."*

Marking notes:

- 0.5 marks for a caption which mentions "1a" and contains the required phrase;
- Inheritance:
  - 0.5 marks for the inheritance of JPanel by HuckleBuckle;
  - 0.5 marks for the inheritance of Point by Player;
- 0.5 marks for the implementation of ActionListener by HuckleBuckle;
- Navigation and Multiplicity:
  - 0.5 marks for a one-way navigable association of a Temperature with each GridCell;
  - 0.5 marks for showing a multiplicity of 1 on the Temperature end of the GridCell-Temperature association;
- Note that this marking scheme is a *spot-check* on the seven inheritances and five associations in the OO design of hbbv2.0. You should not attempt to check for total correctness, however if the student's submission looks reasonably complete and accurate then write "Well done!" in your comments for this question.

Sample answer: Please see Figure 1 (on the next page):

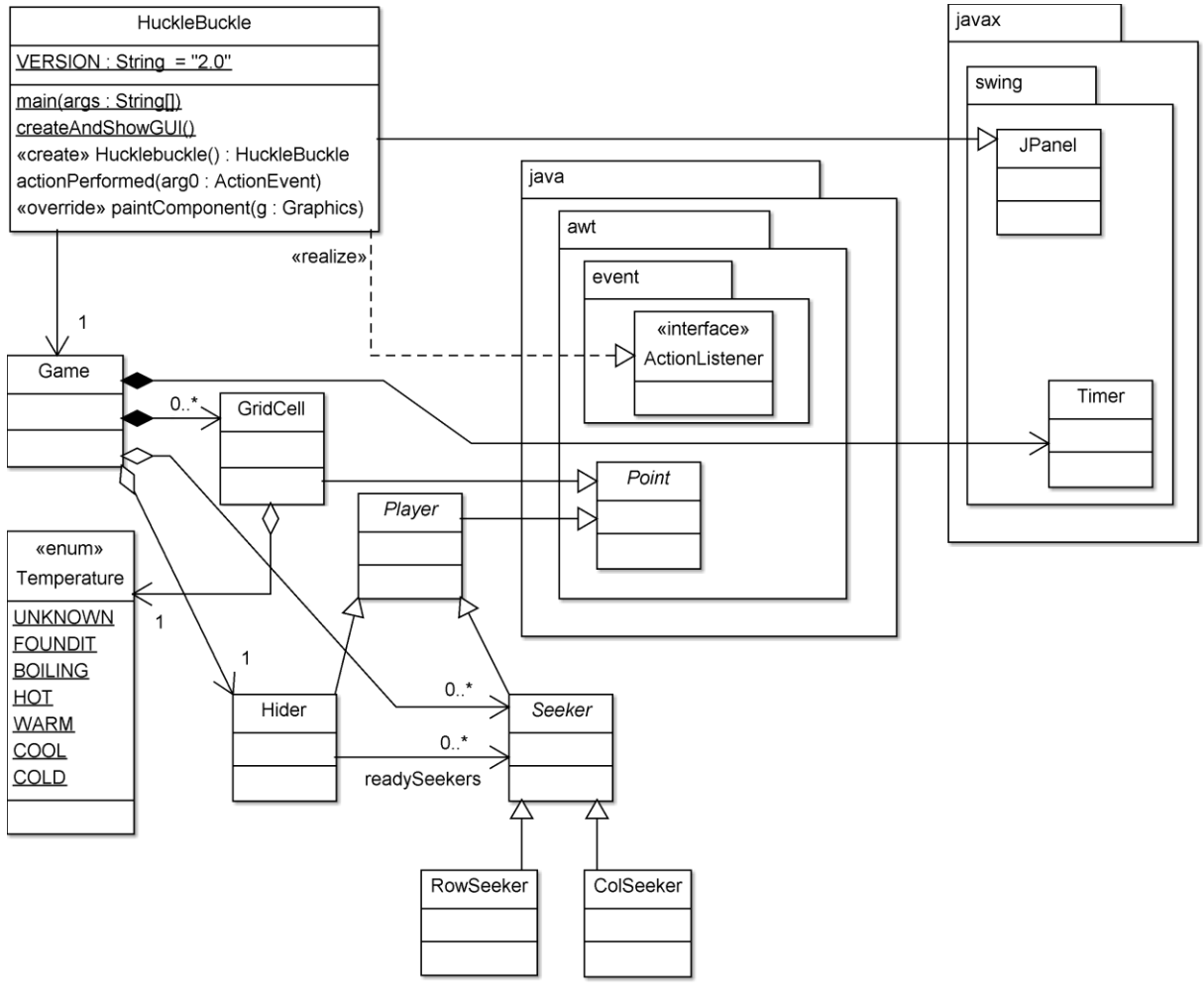


Figure 1. Sample answer to question 1.a, showing the relationships between classes in HuckleBuckle v2.0

b) ***Submit*** *an English-language paragraph, explaining (very briefly) why you specialised one of the associations in your class diagram into an aggregation or composition. To receive full marks, your paragraph must appropriately and understandably use the words "own" and "destroy"; it must clearly indicate which association you are talking about; and it must give some understandable reason "why" you chose to specialise this association. Please also note that English-language syntax is not on our course syllabus, and that our markers are neither trained nor expected to detect, correct, or mark your minor syntax errors in e.g. the conjugation of English verbs (own, owns, owned, …). When answering this question, you are "implementing" English sentences, not Java statements; but in either language, an implementation is not acceptable if it is ambiguous or incomprehensible to its interpreter.*

Sample answer:

I specialised the association between the Timer and the Game into a composition, to indicate that the Timer owned by a Game must be destroyed when the Game is de-allocated. I think this is appropriate semantics because it makes it clear, to the developer, that they should ensure that the Game's Timer is not throwing events after the Game is de-allocated.

Marking notes:

- Keywords:
  - 0.5 marks, for appropriately and understandably using any form of the word "own";
  - 0.5 marks, for appropriately and understandably using any form of the word "destroy";

- 1 mark, for an understandable reason for this specialisation. Note that the semantics of hbbv2.0 are not well-defined in this assignment, so any understandable rationale is acceptable; however if the student's writing is so unclear that you're not even sure which association they're specialising you should award 0 marks. At the other extreme, if a student's explanation is clear and concise, please give them some positive feedback e.g. "Well done!".

Part 2: Learning about the Set interface

2) **(5 marks)** *Assessed work:*

Read the TODO comment in the `HuckleBuckle()` constructor, and also the TODO comment in the Player class. These comments refer to a defect in the behaviour of `hbbv2.0`. The `HuckleBuckle` constructor attempts to add a second Seeker to the set of allSeekers, but this `add()` operation fails – for a reason which you should at least vaguely understand from your initial steps on this problem.

Create `hbbv2.1` with an improved implementation of the `Player` class (but no changes to any other class), so that multiple seekers can play `HuckleBuckle`

Sample answer:

```
abstract class Player extends Point {

    /*
     * Two different Player objects, even if they have the same position and
     * name, are not referring to the same simulated child. Note that we could
     * have two different Seekers named "Sally".
     */
    @Override
    public boolean equals(Object obj) {
        return this == obj;
    }

    /*
     * In an alternative semantics for the HuckleBuckle codebase, we would
     * insist that all (simulated) children have different names. This would
     * allow the developer to construct multiple representations for the same
     * player, and equals() would be defined as below -- two representations
     * refer to the "same player" if they have the same name. Note that the same
     * player, in this semantics, could be in two different places! This would
     * be very confusing, so I have not adopted these alternative semantics for
     * Player-equality.
     */
    public boolean altEquals(Object obj) {
        if (obj instanceof Player) {
            return (getName() == ((Player) obj).getName());
        } else {
            return false;
        }
    }

    /*
     * A third alternative is to define Player-equality by name and position, as
     * below. This is strange semantics, because a developer might instantiate
     * two Players named "Sally" who would be the "same child" whenever they
     * occupy the same grid-square at the same time, but who would be "different
     * children" when they are in different squares.
     */
    public boolean altAltEquals(Object obj) {
```

```
        if (obj instanceof Player) {
            Player pobj = (Player) obj;
            return (super.equals(pobj) && (getName() == pobj.getName()));
        } else {
            return false;
        }
    }

    /*
     * A fourth alternative is to define Player-equality by position. This
     * semantics for player-equality would be appropriate if at most one child
     * can occupy any grid-square at any given time.  Under these semantics, we
     * do not need to override equals(); but this explicit override will
     * facilitate testing, and it also allows us to document the semantics
     * of player-equality in this block comment.
     */
    public boolean altAltAltEquals(Object obj) {
        return super.equals(obj);
    }

}
```

To receive full marks, there should be accurate and informative block-comments on all methods in your listing. There should be no duplication of code, that is, you should call a method rather than duplicating its functionality in another method. Note: you should not attempt to determine the "best" semantics for Player, because there are several "reasonable choices" for an improved Player semantics which would allow Sally to join the game as a Seeker (even though there is already one Seeker in the set of seekers).

Marking notes:

- 2 marks, for a block comment which understandably indicates the student's rationale for their overridden equals() method
- 2 marks, for a plausibly-correct implementation of any of the design options indicated in my sample answer above (or for any other "reasonable" semantics). Note that it would be possible (but rather time-consuming) for a student to modify the methods in Player so that altAltAltEquals() allows multiple seekers to play (by ensuring that there's never more than one Player per GridSquare). The other alternatives require no change to other methods – aside from the removal of the TODO comment in HuckleBuckle's constructor. Ideally the student will list that method!
- 1 marks, for avoidance of duplication e.g. by invoking the equals() method of the superclass as in my altAltEquals() implementation above.

3) ***(6 marks)*** *Assessed work:*

Create hbbv2.1a. This is an "experimental" branch of your `hbbv2.1` *codebase, in which you will try various implementations and types for the* `allSeekers` *variable in the constructor for the* `HuckleBuckle` *class. Note that this is a local variable; you should review slide 21 of the seventh set of lecture slides for a rough idea of the semantics of such variables. Also note that there are just two "uses" of this variable: in the invocation of* `setSeekers()` *of the* `Game` *class, and in the invocation of the* `pleasePlayWith()` *method of the* `Hider` *class. There cannot be any other uses because a local variable isn't accessible outside the method in which it is declared.*

Try the other two implementations for Set in the Collections framework: TreeSet and LinkedHashSet, by calling the relevant constructor in the third line of the HuckleBuckle constructor. (Of course, you'll have to import any package you use!) Do these changes cause any compilation errors, any runtime errors, or any observable change in runtime behaviour?

*Try a few other types for `allSeekers`: it could be declared as a `Collection<Seeker>` rather than a `Set<Seeker>`, it could be declared as a `List<Seeker>`, and it could be declared as a `List`. After you adjust the import statements, are there any compilation errors in any of these three cases? If there were any compilation errors, can you easily repair the defect by making simple changes (such as selecting a more appropriate constructor) to the HuckleBuckle class, or is it necessary to change other classes as well? If you have to change other classes, are these changes "improving" the whole codebase, or are you merely changing multiple classes in order to adapt them to your new HuckleBuckle implementation (with the risk that some future developer might want to "change them again" to suit some other implementation)? After you get each of these three experimental ve**rs**ions to run, determine whether or not there is any observable change in runtime behaviour as a result of your changes to the type of the `allSeekers` reference variable.*

Now review the paragraph in The Set Interface lesson of the Java Tutorials which starts with the phrase "Note that the code always refers to the `Collection` by its interface type…"

***Submit:** a paragraph discussing your experimental findings, and your recommendations (if any) for changing the way the Java Collection Framework is used in hbbv2. Note that you are experimentally evaluating the extensibility of hbbv2, by determining whether or not some "simple" changes to the HuckleBuckle constructor could be made without causing the codebase to "break" (until other classes are changed to suit a changed implementation of HuckleBuckle).*

Sample answer:

My experimental findings were mixed. Shifting to a TreeSet implementation of allSeekers caused a runtime exception, because Seeker doesn't implement the Comparable interface but a TreeSet<S> uses the compare() method of class S to determine the ordering for this set. The stakeholders for this game *might* think it appropriate for the Seekers to be ordered either alphabetically by name, or by the order in which they joined the game; but either of these orderings would be a significant change to the semantics of the game – so shifting to a TreeSet is not merely an implementation decision. Shifting allSeekers to a LinkedHashSet implementation caused no compilation errors (after I imported the relevant package). I couldn't see any difference in runtime efficiency, so I don't think this change would be an improvement. Shifting the type of allSeekers to a Collection<Seeker> caused a compilation error which was easily fixed: I generalised the type of the mySeekers instance variable of Game so that it is a Collection<Seeker> rather than a Set<Seeker>, and (of course) I generalised the signatures of the getter and setter so that they weren't requiring a specialised type of Collection. I'd say this change is a minor improvement to the codebase, because I see no reason the OO designer should insist that mySeekers be implemented as a Set rather than a List or Array. (It is generally considered "good design style" to leave implementation decisions to the implementor, because the resulting design is generally more understandable and extendible.) After generalising the type of the mySeekers instance variable, I was able to shift the type of allSeekers in HuckleBuckle to an Array<Seeker>, with an implementation as an ArrayList<Seeker>, by changing a single line of code. This shift in implementation (to an ArrayList) isn't an improvement, as it makes no observable difference to the code behaviour; but it does demonstrate that the design of the code has indeed been generalised.

Marking notes:

- 1.5 marks, for an understandable discussion of the TreeSet option which mentions the runtime exception.
- 0.5 marks, if the student diagnoses the runtime exception as being caused by a typecast to Comparable, or by an invocation of a compare() method.
- 1 mark, for an understandable discussion of the LinkedHashSet option which indicates there is no observable change in behaviour.

- Collection:
  - 1 mark, for an understandable discussion of the Collection<Seeker> option which indicates the compilation error.
  - 1 mark, for an understandable description of how the codebase could be changed to accommodate the Collection<Seeker> option.
- 1 mark, for an understandable discussion of the List<Seeker> option which indicates that the student was able to compile & run the code after adjusting the implementation of allSeekers (so that it is an Array List<Seeker> or a LinkedList<Seeker>), and after generalising the type of the mySeekers instance variable of Player.

Part 3: An Experiential Introduction to State Diagrams

4) **(6 marks)** *Assessed work: Create a new project called `hbbv2.2`, and import your `hbbv2.1` source code (and not your `hbbv2.1a` experimental code!) into this project. Add a new state, `ASKING`, to the `SeekerState enum`. Adjust the implementation of the Seeker's `move()` method, so that a `Seeker` will enter the `ASKING` state (instead of the `SEEKING` state) after she has taken enough steps to reach her current destination `(nextX, nextY)`. In the `ASKING` state, the seeker should ask the `Hider` for her current temperature. If her temperature isn't `FOUNDIT`, then she should proceed to her (revised) `SEEKING` state – in which she either determines new values for `(nextX, nextY)`, or decides to quit, depending on whether or not she has already looked at all possible hiding places. Note that the effect of this change is to slow down the seeker's progress, for she'll spend two timer-ticks in each cell before moving to the next. This change also makes it possible for a new type of seeker (which you'll develop later in this assignment) to move rapidly through a cell (i.e. without `ASKING` the hider for its temperature) if the temperature is already known.*

Hint: if you're having trouble debugging and you had based your development on hbbv2.0, I'd suggest you consider looking at hbbv2.01. In that code I have correctly implemented the state diagram of Figure 2 in the move() method, I have deleted the moveTo() method, and I have adjusted the output to indicate that a Seeker takes just one step at a time.

***Submit:** a listing of your revised move() method; and a listing of the console output for a run of HuckleBuckle v2.2 with its default arguments. You should not submit listings of the other methods you changed when creating hbbv2.2: you will be marked only on your move() method and your console output. If you are unable to your program to work correctly, you should still submit your revised move() – because the majority of marks are based on this, and not on the output listing.*

Sample answer:

```
/**
 * Invoking this method causes this Seeker to make one move in her current
 * Game.
 *
 * @return true if the Seeker can make any more moves in this game.
 *
 */
public boolean move() {

    switch (myState) {
    case WAITING: // I could play another game if a Hider would invite me...
        // but I won't do anything without an invitation!
        break;
```

```
        case STARTING:
            nextX = 0; // I always look at (0,0) first
            nextY = 0;
            myState = MOVING;
            break;
        case MOVING:
            if ( (getX() == nextX) && (getY() == nextY ) ){
                // I have reached my destination
                myState = ASKING;
            } else {
                // I move one step toward (nextX, nextY)
                translate((int) Math.signum(nextX - getX()),
                        (int) Math.signum(nextY - getY()) );
                distanceMoved++; // Note: moveTo() is no longer called in tryNewPosition()
                System.out.println(getName() + " says, \"I took a step.  I'm now at "
            + (int) getX() + ", " + (int) getY() + ".\"");
                myState = MOVING; // I'm still moving
            }
            break;
        case ASKING:
            System.out.println(getName() + " asks, \"Am I close?\"");
            if (myHider.pleaseRevealTemperatureOf(this) == Temperature.FOUNDIT) {
                myState = WINNING; // Hooray!
            } else {
                myState = SEEKING;
            }
            break;
        case SEEKING:
            if (tryNewPosition()) {  // This method updates (nextX, nextY)
                myState = MOVING; // I keep looking
            } else {
                myState = QUITTING; // I give up!
            }
            break;
        case QUITTING: // I don't move, but I do say something before WAITING.
            System.out.println(getName() + " says, \"I give up!  I took "
                    + distanceMoved + " steps before quitting.\"");
            myState = WAITING;
            break;
        case WINNING: // I don't move, but I do say something before WAITING.
            System.out.println(getName() + " says, \"That was fun! I walked "
                    + distanceMoved + " step" + (distanceMoved != 1 ? "s" : "")
                    + " before I found it.\"");
            myState = WAITING;
            break;
        }
        return (myState != WAITING);
    }
```

```
Playing HuckleBuckle (v2.1) on a 5 by 5 grid...

  Harry says "Hi Sue, I'm Harry, let's play Huckle Buckle!"
Sue says, "Hi, Harry, I'm Sue.  Glad to meet you!  Are you ready?"
  Harry says "Hi Sally, I'm Harry, let's play Huckle Buckle!"
Sally says, "Hi, Harry, I'm Sally.  Glad to meet you!  Are you ready?"
  Harry says to everyone, "I'm ready now.  I bet you can't find it!"
Sue asks, "Am I close?"
  Harry says to Sue, "You're cold."
Sally asks, "Am I close?"
  Harry says to Sally, "You're cold."
Sue says, "I took a step.  I'm now at 0, 1."
Sally says, "I took a step.  I'm now at 1, 0."
```

```
Sue asks, "Am I close?"
  Harry says to Sue, "You're cool."
Sally asks, "Am I close?"
  Harry says to Sally, "You're cool."
Sue says, "I took a step.  I'm now at 0, 2."
Sally says, "I took a step.  I'm now at 2, 0."
Sue asks, "Am I close?"
  Harry says to Sue, "You're warm."
Sally asks, "Am I close?"
  Harry says to Sally, "You're cool."
Sue says, "I took a step.  I'm now at 0, 3."
Sally says, "I took a step.  I'm now at 3, 0."
Sue asks, "Am I close?"
  Harry says to Sue, "You're warm."
Sally asks, "Am I close?"
  Harry says to Sally, "You're cool."
Sue says, "I took a step.  I'm now at 0, 4."
Sally says, "I took a step.  I'm now at 4, 0."
Sue asks, "Am I close?"
  Harry says to Sue, "You're warm."
Sally asks, "Am I close?"
  Harry says to Sally, "You're cold."
Sue says, "I took a step.  I'm now at 1, 4."
Sally says, "I took a step.  I'm now at 4, 1."
Sue asks, "Am I close?"
  Harry says to Sue, "You're hot."
Sally asks, "Am I close?"
  Harry says to Sally, "You're cool."
Sue says, "I took a step.  I'm now at 1, 3."
Sally says, "I took a step.  I'm now at 3, 1."
Sue asks, "Am I close?"
  Harry says to Sue, "You're boiling!"
Sally asks, "Am I close?"
  Harry says to Sally, "You're warm."
Sue says, "I took a step.  I'm now at 1, 2."
Sally says, "I took a step.  I'm now at 2, 1."
Sue asks, "Am I close?"
  Harry says to Sue, "You're hot."
Sally asks, "Am I close?"
  Harry says to Sally, "You're warm."
Sue says, "I took a step.  I'm now at 1, 1."
Sally says, "I took a step.  I'm now at 1, 1."
Sue asks, "Am I close?"
  Harry says to Sue, "You're warm."
Sally asks, "Am I close?"
  Harry says to Sally, "You're warm."
Sue says, "I took a step.  I'm now at 1, 0."
Sally says, "I took a step.  I'm now at 0, 1."
Sue asks, "Am I close?"
  Harry says to Sue, "You're cool."
Sally asks, "Am I close?"
  Harry says to Sally, "You're cool."
Sue says, "I took a step.  I'm now at 2, 0."
Sally says, "I took a step.  I'm now at 0, 2."
Sue asks, "Am I close?"
  Harry says to Sue, "You're cool."
Sally asks, "Am I close?"
  Harry says to Sally, "You're warm."
Sue says, "I took a step.  I'm now at 2, 1."
Sally says, "I took a step.  I'm now at 1, 2."
Sue asks, "Am I close?"
  Harry says to Sue, "You're warm."
Sally asks, "Am I close?"
```

```
  Harry says to Sally, "You're hot."
Sue says, "I took a step.  I'm now at 2, 2."
Sally says, "I took a step.  I'm now at 2, 2."
Sue asks, "Am I close?"
  Harry says to Sue, "You're boiling!"
Sally asks, "Am I close?"
  Harry says to Sally, "You're boiling!"
Sue says, "I took a step.  I'm now at 2, 3."
Sally says, "I took a step.  I'm now at 3, 2."
Sue asks, "Am I close?"
  Harry says to Sue, "Huckle buckle beanstalk!"
Sally asks, "Am I close?"
  Harry says to Sally, "You're hot."
Sue says, "That was fun! I walked 13 steps before I found it."
Sally says, "I took a step.  I'm now at 4, 2."
Sally asks, "Am I close?"
  Harry says to Sally, "You're warm."
Sally says, "I took a step.  I'm now at 4, 3."
Sally asks, "Am I close?"
  Harry says to Sally, "You're warm."
Sally says, "I took a step.  I'm now at 3, 3."
Sally asks, "Am I close?"
  Harry says to Sally, "You're boiling!"
Sally says, "I took a step.  I'm now at 2, 3."
Sally asks, "Am I close?"
  Harry says to Sally, "Huckle buckle beanstalk!"
Sally says, "That was fun! I walked 17 steps before I found it."
```

Marking notes:

- 2 marks, for a move() method that has an ASKING case which tests `myHider.pleaseRevealTemperatureOf(this);`
- 2 marks, for a move() method that has a SEEKING case which tests `tryNewPosition()` to determine whether the next state will be MOVING or QUITTING;
- 2 marks, for an output trace showing a plausible run of the modified program.

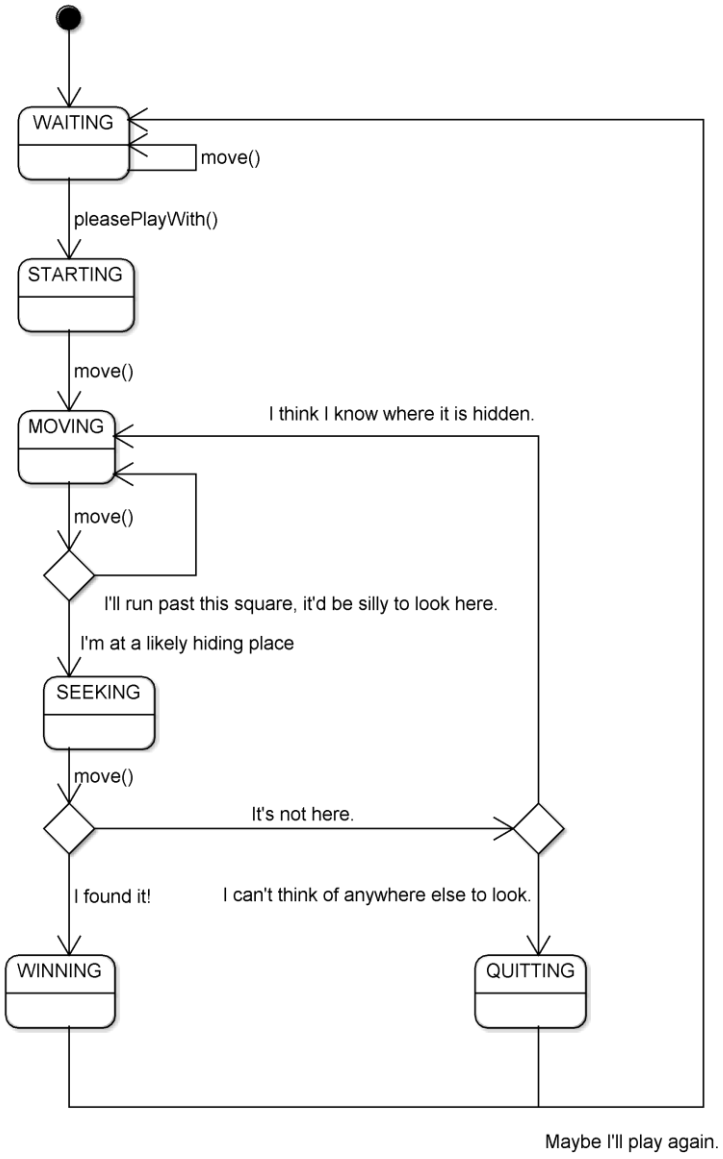


Figure 2. State diagram for Seeker in hbbv2.0.

Part 4: Adding Three Buttons

5) ***(6 marks)*** *Assessed work. Create a new project, import your previous version of the* `hbbv2.x` *codebase, and modify this code until it displays three buttons. One should be labelled "Pause", one should be labelled "Resume", and one should be labelled "Single Step". The action handlers for these buttons should not actually pause, resume or single-step the game; however you should*

- *place these buttons along the top edge of the* `JPanel` *(with the grid displayed below),*
- *assign appropriate labels to events (e.g. "pause" for the Pause button)*
- *assign appropriate keyboard shortcuts (aka mnemonics), e.g. "P" for the Pause button. (Note that, by convention, an underlined letter in a menu or button label indicates the keystroke which will invoke this button; this demo is defective in that it responds to an* `<Alt>p` *but not an* `<Alt>P`*, even though the "P" is capitalised in the "Pause" label on its Pause button.)*
- *ensure that the Resume button is initially disabled, and is enabled after Pause is clicked*
- *ensure that that the Pause button is initially enabled, and is disabled after Pause is clicked*

When answering the next problem, you'll be modifying the button enable/disable behaviour, so you should not attempt to develop appropriate button-click behaviour (aside from what is specified above) in this version of your code.

***Submit:** a listing of any method you modified or added, when implementing the features described above, and two screenshots showing the appearance of your app's GUI window before, and after, the Pause is clicked for the first time.*

Sample answer:

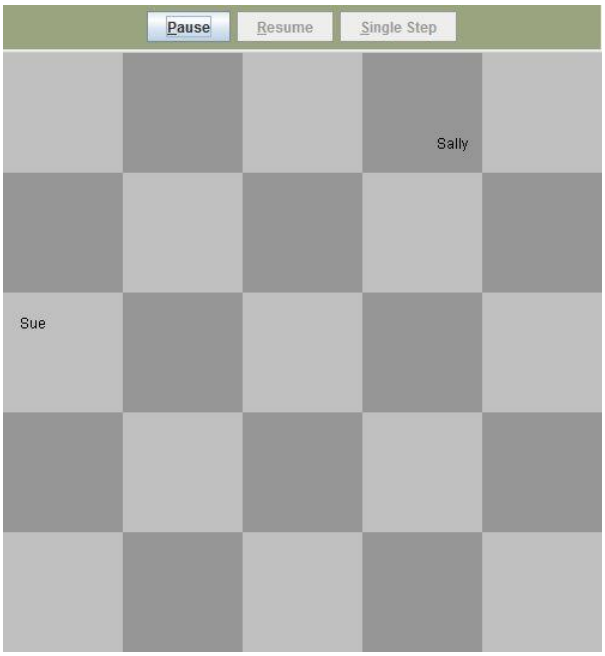
I modified the createAndShowGUI() of HuckleBuckle, and I added a new class HbbButtons, as listed below.

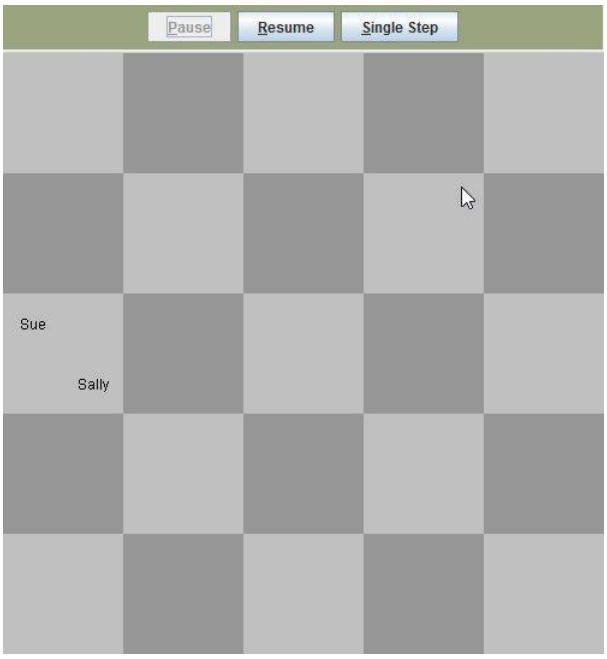
```
    /**
     * Create the GUI and show it.  For thread safety,
     * this method should be invoked from the
     * event-dispatching thread.
     */
    private static void createAndShowGUI() {
        //Create and set up the window.
        JFrame frame = new JFrame("HuckleBuckle viewer");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setPreferredSize(new Dimension(500, 500));

        //Create a menu bar with three buttons.  Make it have a green background.
        JMenuBar greenMenuBar = new JMenuBar();
        greenMenuBar.setOpaque(true);
        greenMenuBar.setBackground(new Color(154, 165, 127));
        greenMenuBar.setPreferredSize(new Dimension(120, 40));
        greenMenuBar.add(new HbbButtons(greenMenuBar.getBackground()));

        frame.setJMenuBar(greenMenuBar);
        frame.getContentPane().add(new HuckleBuckle(), BorderLayout.CENTER);

        //Display the window.
        frame.pack(); // it'll be just big enough for the preferred sizes of its components
        frame.setVisible(true);
    }


/*
 * This class is derived from ButtonDemo.java in the Java Tutorials
 *
 * I have overridden paintComponent(), so that these buttons will honour the
 * background colour of the JMenuBar in which they are embedded.  Without
 * the override, the background colour may be defined by the look-and-feel, see
 * https://docs.oracle.com/javase/8/docs/api/javax/swing/JComponent.html#setBackground-
java.awt.Color-
 *
 */
@SuppressWarnings("serial")
public class HbbButtons extends JPanel
                        implements ActionListener {

    protected JButton bPause, bResume, bSingleStep;
    private Color backgroundColour;

    /**
     * Creates three buttons, and adds them to this JPanel
     *
     * @param bg
     *      background colour
```

```
     */
    public HbbButtons(Color bg) {

        backgroundColour = bg;

        bPause = new JButton("Pause", null);
        bPause.setVerticalTextPosition(AbstractButton.CENTER);
        bPause.setHorizontalTextPosition(AbstractButton.LEADING); //aka LEFT, for left-to-right
locales
        bPause.setMnemonic(KeyEvent.VK_P);
        bPause.setActionCommand("pause");
        bPause.setOpaque(true);

        bResume = new JButton("Resume", null);
        bResume.setVerticalTextPosition(AbstractButton.BOTTOM);
        bResume.setHorizontalTextPosition(AbstractButton.CENTER);
        bResume.setMnemonic(KeyEvent.VK_R);
        bResume.setActionCommand("resume");
        bResume.setEnabled(false);
        bResume.setOpaque(true);

        bSingleStep = new JButton("Single Step", null);
        //Use the default text position of CENTER, TRAILING (RIGHT).
        bSingleStep.setMnemonic(KeyEvent.VK_S);
        bSingleStep.setActionCommand("step");
        bSingleStep.setEnabled(false);
        bSingleStep.setOpaque(true);

        //Listen for actions
        bPause.addActionListener(this);
        bResume.addActionListener(this);
        bSingleStep.addActionListener(this);

        bPause.setToolTipText("Click this button to pause the simulation.");
        bResume.setToolTipText("Click this button to resume the simulation.");
        bSingleStep.setToolTipText("Click this button to allow a single move.");

        //Add Components to this container, using the default FlowLayout.
        add(bPause);
        add(bResume);
        add(bSingleStep);
    }

    /*
     * The game's button events are handled here
     */
    public void actionPerformed(ActionEvent e) {
    switch (e.getActionCommand()) {
        case "pause":
            bPause.setEnabled(false);
            bResume.setEnabled(true);
            bSingleStep.setEnabled(true);
            break;
        case "resume":
            bPause.setEnabled(true);
            bResume.setEnabled(false);
            bSingleStep.setEnabled(false);
            break;
        case "step":
        }
    }

    @Override
```

```
    protected void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        g.setColor(backgroundColour);
        g.fillRect(0, 0, getWidth() - 1, getHeight() - 1);
    }
}
```

Screenshots (before and after a Pause):





Marking notes:

- 2 marks, for displaying three buttons anywhere in the viewing screen;
- Labels:
  - 1 mark, for three appropriate event labels e.g. “pause” in three `setActionCommand()` invocations;
  - 1 mark, for three appropriate mnemonics e.g. VK_P, in `setMnemonic()` invocations;
- Disable/enable:
  - 1 mark, for initially disabling Resume and enabling Pause;
  - 1 mark, for disabling Pause after it is clicked, and enabling Resume.

Write “well done!” in the comments field if the student has nicely integrated or “cleaned up” the code from JButtonDemo, e.g. by labelling the buttons mnemonically (rather than b1, b2, b3), or by using a switch statement (instead of an if-else-else construct) in the `actionPerformed()` method.

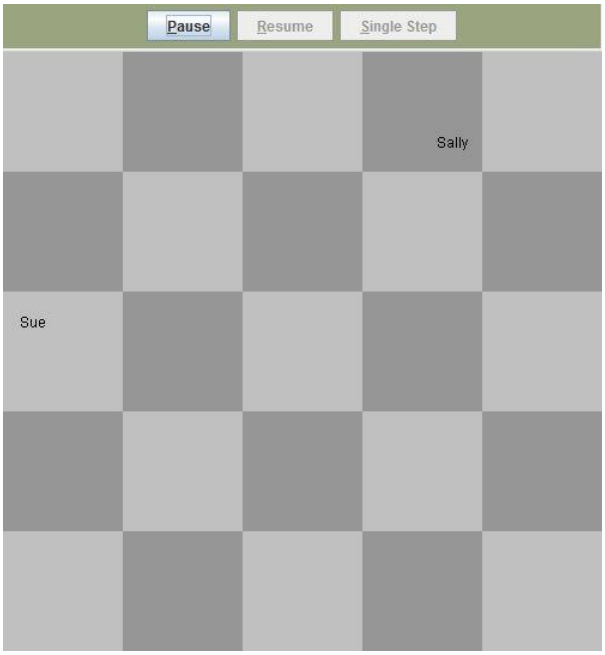
6) **(6 marks)** *Assessed work.* Invoke the stop(), start(), and setRepeats(false) methods on the game timer, in the actionPerformed() handler for your buttons, so that your buttons have the appropriate behaviour of pausing, resuming, and single-stepping the simulation of a game of Huckle Buckle Beanstalk. To receive full marks, your app should disable any button which shouldn’t be pressed, e.g. the Resume and Single-Step buttons should be disabled when the simulation is running.

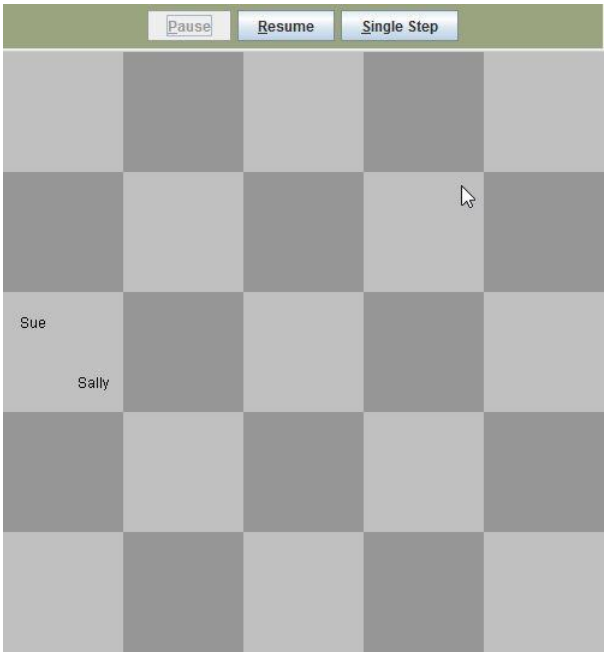
Submit: a listing of any method you modified or added, when implementing the features described above, and two screenshots showing the appearance of your app’s GUI window before, and after, the Pause is clicked for the first time.

Sample answer: I modified the `actionPerformed()` method of HbbButtons, and I added diagnostic ("Tick, Tock") output to the `actionPerformed()` method of HuckleBuckle, as listed below.

```
/*
 * The game's button events are handled here
 */
public void actionPerformed(ActionEvent e) {
switch (e.getActionCommand()) {
    case "pause":
        bPause.setEnabled(false);
        bResume.setEnabled(true);
        bSingleStep.setEnabled(true);
        HuckleBuckle.myGame.getTimer().stop();
        break;
    case "resume":
        bPause.setEnabled(true);
        bResume.setEnabled(false);
        bSingleStep.setEnabled(false);
        HuckleBuckle.myGame.getTimer().setRepeats(true);
        HuckleBuckle.myGame.getTimer().start();
        break;
    case "step":
        HuckleBuckle.myGame.getTimer().setRepeats(false);
        HuckleBuckle.myGame.getTimer().start();
    }
}

/*
 * The game's timer events are handled here
 */
static boolean isTick;
@Override
public void actionPerformed(ActionEvent arg0) {
    System.out.println("    " + (isTick ? "Tick" : "Tock") );
    isTick = !isTick;
    for (Seeker s : myGame.getSeekers()) {
        s.move(); // change the model, to reflect the passage of time
        repaint(); // update the view, for consistency with the model
    }
}
```

Screenshots (before and after a Pause):





Marking notes:

- 2 marks, for invoking the game timer's stop() method in the actionEvent() hooked to the Pause button;
- Resume:
  - 1 mark, for setting the setRepeats() flag in the actionEvent() hooked to the Resume button;
  - 1 mark, for starting the game's timer in the actionEvent() hooked to the Resume button;
- Single Step:
  - 1 mark, for clearing the setRepeats() flag in the actionEvent() hooked to the SingleStep button prior to starting or restarting the timer;
  - 1 mark, for starting or restarting the game's timer in the actionEvent() hooked to the SingleStep button. Note that some students may add additional logic to the single-step affordance, so that the seekers take one visible step (generally two timer-ticks) every time the single-step button is depressed. If a student succeeds in implementing this logic, write "Well done, that's a very appropriate implementation of single-stepping for this application."

Part 5: Adding Instance Variables and Methods to enums, and Custom Painting

7) **(6 marks)** *Assessed work. Develop a new version of Huckle Buckle, by modifying the* `Temperature enum` *in your previous version so that it associates a* `Color` *with each instance of the* `enum`*, and by modifying the custom-painting code in HuckleBuckle so that it paints a temperature-appropriate color on each square after Harry has revealed its temperature to either of the seekers.* ***Submit*** *a listing of your* `Temperature enum`*, a listing of any custom-painting method that you modified (or added), a listing of the method(s) you modified or added when recording the hider's temperature reports in the Game object, and a screenshot showing at least four different colors for GridCells.*

Sample answer:

Temperature enum:

```
package hucklebuckle;

import java.awt.Color;

public enum Temperature {
    UNKNOWN(Color.LIGHT_GRAY),
    FOUNDIT(Color.WHITE),
    BOILING(Color.RED),
    HOT(Color.ORANGE),
    WARM(Color.YELLOW),
    COOL(Color.GREEN),
    COLD(Color.BLUE),
    FREEZING(Color.DARK_GRAY);

    private final Color myColor;

    Temperature(Color c){
        myColor = c;
    }

    Color getColor() {
        return myColor;
    }

}
```

Modified painting method:

```
    /*
     * Paints the game-grid and the seekers.
     *
     */
    @Override
    protected void paintComponent(Graphics g) {

        super.paintComponent(g);

        // A HuckleBuckle instance is-a JPanel, so it has a display area
        // in a GUI window which may be resized. How big is it now?
        int width = getSize().width;
        int height = getSize().height;

        // Grid cells are square, even when the display area is rectangular
        int cellSize = Math.min(width, height) / myGame.getGridSize();

        // Display grid in checkerboard gray, initially.
        Color lighterGray = new Color(150, 150, 150);
        for (GridCell gc : myGame.getGrid()) {
            if (gc.getTemperature() == Temperature.UNKNOWN
                    && (((gc.x % 2) ^ (gc.y % 2)) == 0)) {
                g.setColor(lighterGray);
            } else {
                g.setColor(gc.getTemperature().getColor());
            }

            // Note: I scale the cell indices (gc.x, gc.y) by cellSize,
            // so that cells are of an appropriate size for our current
            // view-window.
            g.fillRect(gc.x * cellSize, gc.y * cellSize, cellSize, cellSize);
        }
```

```
        // display Seeker names in black texxt
        g.setColor(Color.BLACK);

        int numSeekers = myGame.getSeekers().size();

        // Offset seeker names within each grid cell, to avoid overlap
        int xOffset = 0;
        int yOffset = 0;
        int deltaOffset = cellSize / numSeekers;

        for (Seeker s : myGame.getSeekers()) {
            // The semantics of font-rendering in AWT is not examinable!
            // Here I'm calculating margins for a string-display within a cell.
            int xMargin = (deltaOffset - g.getFontMetrics().stringWidth(
                    s.getName())) / 2;
            int yMargin = (deltaOffset - g.getFontMetrics().getHeight()) / 2
                    + g.getFontMetrics().getAscent();
            // Note: I scale the cell indices (s.x, s.y) by cellSize,
            // so that the seeker-strings are displayed inside their cell.
            g.drawString(s.getName(), s.x * cellSize + xOffset + xMargin, s.y
                    * cellSize + yOffset + yMargin);
            // Seeker names are offset diagonally.
            xOffset += deltaOffset;
            yOffset += deltaOffset;
        }
    }
```

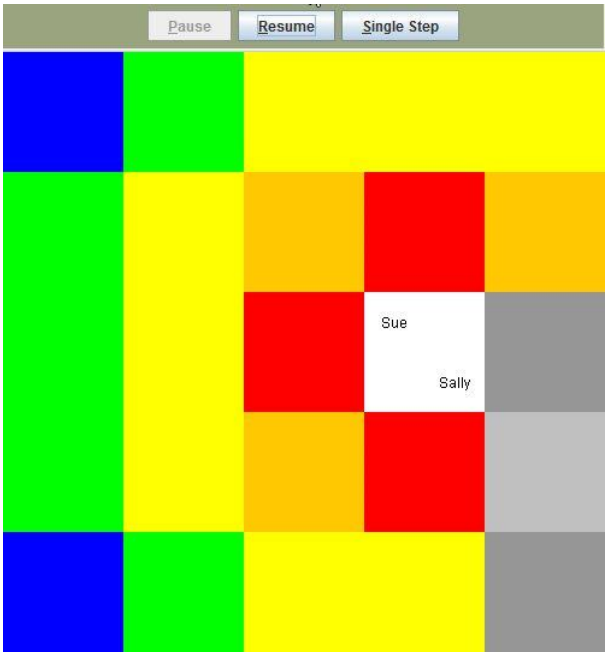
Adding a getCell() method to Game:

```
    /**
     * getCell(x,y) returns the GridCell at location (x,y)
     *
     * TODO: implement myGrid as an ArrayList<ArrayList<GridCell>>, to avoid
     * writing low-level array-indexing arithmetic in this routine.  Alternatively,
     * build a HashMap<Point,GridCell> when adding GridCells, so that they
     * can be indexed efficiently.  A third possibility would be to implement
     * myGrid as a two-dimensional array of GridCells, and to convert from
     * that representation into a List<GridCell> when exporting via getGrid().
     * A fourth possibility would be to adjust the signature of getGrid() so that
     * it returns a two-dimensional array, but this would export an implementation
     * decision to the entire codebase so (unless there were some compelling reason
     * for this export) I wouldn't choose that design option.
     */
    GridCell getCell(int x, int y){
        return myGrid.get(x*getGridSize() + y);
    }
```

Recording temperatures (by adjusting one case in the Seeker's move() method):

```
        case ASKING:
            System.out.println(getName() + " asks, \"Am I close?\"");
            Temperature myTemp = myHider.pleaseRevealTemperatureOf(this);
            getGame().getCell((int) getX(), (int) getY()).setTemperature(myTemp);
            if (myTemp == Temperature.FOUNDIT) {
                myState = WINNING; // Hooray!
            } else {
                myState = SEEKING;
            }
            break;
```

Screenshot:



Marking notes:

- 1 mark, for writing a constructor for a Temperature enum that has a Color field,
- 1 mark, for writing a getter for the Color field of the Temperature enum,
- 1 mark, for choosing appropriate Color constants,
- 3 marks, for painting colors on the display. This should be all-or-nothing marking, because there are a wide variety of ways in which a student could adjust the Game class to provide random-access to GridCells by their (x,y) location.

Part 6: Should a Player have a Point, or should she be a Point?

8) ***(4 marks)*** *Assessed work: Consider the appropriateness of a significant change to the Huckle Buckle codebase: a Player might have-a Point rather than being-a Point. As discussed in lecture, it is generally a bad idea to use inheritance, when a composition is feasible. Explore this design decision, by creating a new version of Huckle Buckle in which a Player has-a Point. Keep a record of the lines of code you change, when creating this new version.* ***Submit:*** *a paragraph evaluating this design change: does composition (of a Point, when defining a Player) have any significant advantages or disadvantages over inheritance? Also submit a paragraph discussing any significant difficulties (such as a large number of code-changes) you encountered, when shifting this inheritance to a composition.*

Sample answer:

I see a couple of minor advantages, and no significant disadvantages, in using composition over inheritance in this case. Composition seems more appropriate semantically, because I prefer to think of the players as children who occupy various GridCells as their game progresses – they are not extensions of display objects, but they can be displayed, and composition reflects this. Composition also leads to a somewhat "cleaner" implementation, as discussed in the next paragraph.

The implementation wasn't changed dramatically, but it was improved slightly when I adjusted the code to implement the design change. My first step was to add getters and setters to Player: getPoint(), setPoint(), getX(), setX(), getY(), and setY(). My getX() and getY() methods return an int, because the grid cells have integer-valued

coordinates; this allows a minor implementation advantage over inheritance, because the getX() and getY() methods of a Point return a double which is cast to an integer (or must be formatted) before printing. It is not possible to overload the getX() and setX() methods of Point by writing new int-valued methods in Player, so this is a case where an inheritance was unnecessarily limiting the implementer's range of freedom, forcing them to write somewhat awkward code to work around a design constraint. Aside from the new getters and setters, the definition of a Point-valued instance variable in Player, and a minor adjustment to its constructor, I had to change only a few lines of code in other classes. The only affected lines were ones in which a Point-specific method was invoked on a Player. These lines were easily modified by inserting a getPoint() invocation. I think the resulting code is slightly preferable, even though the additional method invocation makes it slightly more verbose, because the Point-specific methods are now clearly distinguished from the methods which are specific to a Player. Recoding for this shift in design had one other beneficial effect: it exposed a defect in the implementation of the paintComponent() method of HuckleBuckle of hbbv2.0. The hbbv2.0 implementation referred to the (non-private) fields x and y of a Seeker, but it really should have be using the getX() and getY() methods -- to avoid an unnecessary dependence on the implementation of these fields. I repaired this defect when rewriting this line of code to

```
g.drawString(s.getName(), s.getX() * cellSize + xOffset + xMargin, s.getY()
        * cellSize + yOffset + yMargin);
```

Marking notes:

- 2 marks, for an understandable paragraph on the topic of the design principle of 'prefer composition over inheritance' in the context of the Player class in the HuckleBuckle codebase. To receive full marks, the student must identify at least one advantage or disadvantage, and they must refer to something specific about this design (rather than writing generally about a design theory).
- 2 marks, for an understandable paragraph on the topic of implementing this change in design. To receive full marks, the student must describe at least one "coding challenge" they successfully overcame during this implementation. Note that a student need not completely implement the change in design to receive full marks, but they are expected to assess its appropriateness so they must (at minimum) develop an estimate of the work involved (e.g. a rough count of the number of lines of code that must be changed).

Part 7: Only one seeker per GridCell!

9) ***(6 marks)*** *Assessed work: Adjust the "rules of the game" so that, with the exception of the initial cell (0,0), a seeker is not allowed to execute its translate() method if this would put it into the same cell as some other seeker. A seeker can thus get "stuck" in its MOVING state, until another seeker moves out of the way; and there can be only one winner in a game (unless the object is hidden in the initial cell).*

- *You **should** add a counter to the MOVING state, so that a seeker will enter the QUITTING state if she can't move for an extended period of time (say, 8 timer-events).*
- *You **should** test your program with many seekers; so you are required to add a second command-line argument which defines the number of seekers (default two; maximum 10; minimum one).*
- *You **should** add a new type of seeker, one which "looks ahead" when picking her next goal (nextX, nextY) – she shouldn't enter an ASKING state on any cell whose temperature is already known. You will not be marked on any additional "cleverness" you add to this seeker, so I'd encourage you not to get "too clever" until you have completed a version with the required features.*

- *Seekers **should** be of diverse types in a multi-seeker game, so your app should instantiate your clever seeker first, then a row-seeker, then a col-seeker, and repeating this sequence until it has instantiated the number of seekers specified by the command-line argument.*
- *Seekers **should** output informative messages, so that the console output is diagnostic of their current position and state.*
- *You **may** base this development on the code you developed before starting question 8, that is, your Players may be Points, rather than having a Point.*

- ***Submit:***
  - *A listing of the move() method of your modified Seeker,*
  - *A listing of the tryNewPosition() method of your "clever" Seeker subtype, and*
  - *A sample of the console and GUI output of your final codebase, for some interesting set of commandline arguments (e.g. perhaps with four seekers on a 7x7 grid), with the snapshot of your GUI being taken at some interesting time in the simulation.*

*In addition to your written answers (as specified above), you should also **submit** a jarfile containing your source code for your final version.*

Sample answer:

Move() method of Seeker:

```
/**
 * Invoking this method causes this Seeker to make one move in her current
 * Game.
 *
 * @return true if the Seeker can make any more moves in this game.
 *
 */
public boolean move() {

    switch (myState) {
    case WAITING: // I could play another game if a Hider would invite me...
        // but I won't do anything without an invitation!
        break;
    case STARTING:
        nextX = 0; // I always look at (0,0) first
        nextY = 0;
        quitCounter = 0;
        myState = MOVING;
        break;
    case MOVING:
        if ((getX() == nextX) && (getY() == nextY)) {
            // I have reached my destination
            myState = ASKING;
        } else {
            // Is there already a seeker in the GridCell I will enter next?
            int nextCellX = getX() + (int) Math.signum(nextX - getX());
            int nextCellY = getY() + (int) Math.signum(nextY - getY());
            if (getGame().getCell(nextCellX, nextCellY).getSeekers().size() > 0) {
                if (++quitCounter > QUIT_COUNTER_LIMIT) {
                    System.out.println(getName()
                            + " says, \"I'm tired of waiting!\"");
                    myState = QUITTING;
                } else {
                    System.out.println(getName()
```

```
                                + " says, \"I'm trying to move to " + nextCellX
                                + ", " + nextCellY + ".\"");
                        myState = MOVING; // I'll wait until they get out of my
                                          // way
                    }
                } else {
                    // I move one step toward (nextX, nextY).
                    getGame().getCell((int) getX(), (int) getY()).getSeekers()
                            .remove(this);
                    getPoint().setLocation(nextCellX, nextCellY);
                    getGame().getCell((int) getX(), (int) getY()).getSeekers()
                            .add(this);
                    distanceMoved++;
                    quitCounter = 0;
                    System.out.println(getName()
                            + " says, \"I took a step.  I'm now at "
                            + (int) getX() + ", " + (int) getY() + ".\"");
                    myState = MOVING; // I'm still moving
                }
            }
            break;
        case ASKING:
            System.out.println(getName() + " asks, \"Am I close?\"");
            Temperature myTemp = myHider.pleaseRevealTemperatureOf(this);
            getGame().getCell((int) getX(), (int) getY())
                    .setTemperature(myTemp);
            if (myTemp == Temperature.FOUNDIT) {
                myState = WINNING; // Hooray!
            } else {
                myState = SEEKING;
            }
            break;
        case SEEKING:
            if (tryNewPosition()) { // This method updates (nextX, nextY)
                quitCounter = 0;
                myState = MOVING; // I keep looking
            } else {
                myState = QUITTING; // I give up!
            }
            break;
        case QUITTING: // I don't move, but I do say something before WAITING.
            System.out.println(getName() + " says, \"I give up!  I took "
                    + distanceMoved + " steps before quitting.\"");
            myState = WAITING;
            break;
        case WINNING: // I don't move, but I do say something before WAITING.
            System.out.println(getName() + " says, \"That was fun! I walked "
                    + distanceMoved + " step" + (distanceMoved != 1 ? "s" : "")
                    + " before I found it.\"");
            myState = WAITING;
            break;
        }
        return (myState != WAITING);
    }
```

tryNewPosition() method of my clever seeker:

```
/**
     * I move right on even-numbered rows, then left on odd-numbered rows
     *
     * @return false if I don't move (because I have already looked everywhere,
     *         or because all of the cells in my snakelike path are either
```

```
     *         occupied or have a known temperature). Note that I can never win
     *         by moving into a cell that has already been occupied, unless that
     *         seeker doesn't ask about it's temperature. However I *could* be
     *         much more clever if I moved toward HOT and away from COLD...
     *
     */
    @Override
    boolean tryNewPosition() {
        nextX = (int) getX(); // (nextX, nextY) will be my next destination
        nextY = (int) getY();
        // I look for an empty destination with an unknown temperature
        do {
            if ((nextY % 2) == 0) { // I move right on even-numbered rows
                nextX++;
                if (nextX >= getGame().getGridSize()) {
                    nextX--;
                    nextY++; // I move to the next row
                }
            } else { // I move left on odd-numbered rows
                nextX--;
                if (nextX < 0) { // Am I past the end of a row?
                    nextX++; // Yes, so I shouldn't move left
                    nextY++; // I move to the next row
                }
            }
            if (nextY >= getGame().getGridSize()) {
                nextY--; // I don't move off the last row
                return false; // I can't move
            }
        } while (getGame().getCell(nextX, nextY).getTemperature() != Temperature.UNKNOWN
                || getGame().getCell(nextX, nextY).getSeekers().size() != 0);
        return true;
    }
```

Notes on my console output and screenshot:

My two “clever” seekers (Salah, Sue) not very smart – they can’t figure out that the seeker who is occupying the cell they’re currently trying to enter may also be waiting for someone to move. When I paused this simulation to take the screenshot and console listing, Salah was on 5,1 and Sally was on 6,1 – they were trying to occupy each other’s cell so neither could move. Sue was on 5,0 and was also trying to enter 6,1. Only the ColSeeker Sara was able to move.

Console listing:

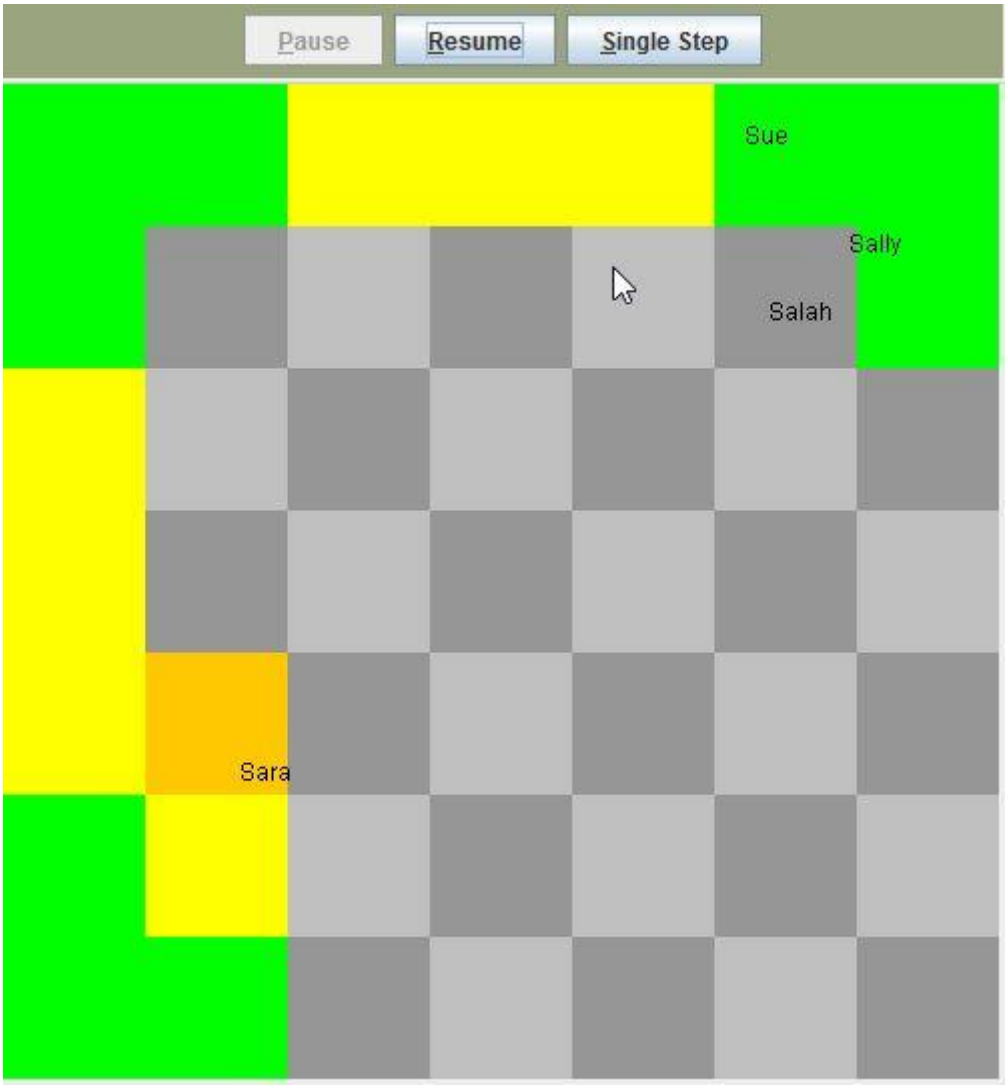
```
Playing HuckleBuckle (v2.7) on a 7 by 7 grid with 4 seekers...

  Harry says "Hi Sally, I'm Harry, let's play Huckle Buckle!"
Sally says, "Hi, Harry, I'm Sally.  Glad to meet you!  Are you ready?"
  Harry says "Hi Sue, I'm Harry, let's play Huckle Buckle!"
Sue says, "Hi, Harry, I'm Sue.  Glad to meet you!  Are you ready?"
  Harry says "Hi Salah, I'm Harry, let's play Huckle Buckle!"
Salah says, "Hi, Harry, I'm Salah.  Glad to meet you!  Are you ready?"
  Harry says "Hi Sara, I'm Harry, let's play Huckle Buckle!"
Sara says, "Hi, Harry, I'm Sara.  Glad to meet you!  Are you ready?"
  Harry says to everyone, "I'm ready now.  I bet you can't find it!"
    Tock
    Tick
    Tock
Sally asks, "Am I close?"
  Harry says to Sally, "You're cool."
```

```
Sue asks, "Am I close?"
  Harry says to Sue, "You're cool."
Salah asks, "Am I close?"
  Harry says to Salah, "You're cool."
Sara asks, "Am I close?"
  Harry says to Sara, "You're cool."
    Tick
    Tock
Sally says, "I took a step.  I'm now at 1, 0."
Sue says, "I'm trying to move to 1, 0."
Salah says, "I'm trying to move to 1, 0."
Sara says, "I took a step.  I'm now at 0, 1."
    Tick
Sue says, "I'm trying to move to 1, 0."
Salah says, "I'm trying to move to 1, 0."
    Tock
Sally asks, "Am I close?"
  Harry says to Sally, "You're cool."
Sue says, "I'm trying to move to 1, 0."
Salah says, "I'm trying to move to 1, 0."
Sara asks, "Am I close?"
  Harry says to Sara, "You're cool."
    Tick
Sue says, "I'm trying to move to 1, 0."
Salah says, "I'm trying to move to 1, 0."
    Tock
Sally says, "I took a step.  I'm now at 2, 0."
Sue says, "I took a step.  I'm now at 1, 0."
Salah says, "I'm trying to move to 1, 0."
Sara says, "I took a step.  I'm now at 0, 2."
    Tick
Salah says, "I'm trying to move to 1, 0."
    Tock
Sally asks, "Am I close?"
  Harry says to Sally, "You're warm."
Sue asks, "Am I close?"
  Harry says to Sue, "You're cool."
Salah says, "I'm trying to move to 1, 0."
Sara asks, "Am I close?"
  Harry says to Sara, "You're warm."
    Tick
Salah says, "I'm trying to move to 1, 0."
    Tock
Sally says, "I took a step.  I'm now at 3, 0."
Sue says, "I took a step.  I'm now at 2, 0."
Salah says, "I took a step.  I'm now at 1, 0."
Sara says, "I took a step.  I'm now at 0, 3."
    Tick
Sue says, "I'm trying to move to 3, 0."
    Tock
Sally asks, "Am I close?"
  Harry says to Sally, "You're warm."
Sue says, "I'm trying to move to 3, 0."
Salah asks, "Am I close?"
  Harry says to Salah, "You're cool."
Sara asks, "Am I close?"
  Harry says to Sara, "You're warm."
    Tick
Sue says, "I'm trying to move to 3, 0."
    Tock
Sally says, "I took a step.  I'm now at 4, 0."
Sue says, "I took a step.  I'm now at 3, 0."
Salah says, "I took a step.  I'm now at 2, 0."
```

```
Sara says, "I took a step.  I'm now at 0, 4."
    Tick
Salah says, "I'm trying to move to 3, 0."
    Tock
Sally asks, "Am I close?"
  Harry says to Sally, "You're warm."
Sue asks, "Am I close?"
  Harry says to Sue, "You're warm."
Salah says, "I'm trying to move to 3, 0."
Sara asks, "Am I close?"
  Harry says to Sara, "You're warm."
    Tick
Salah says, "I'm trying to move to 3, 0."
    Tock
Sally says, "I took a step.  I'm now at 5, 0."
Sue says, "I took a step.  I'm now at 4, 0."
Salah says, "I took a step.  I'm now at 3, 0."
Sara says, "I took a step.  I'm now at 0, 5."
    Tick
Sue says, "I'm trying to move to 5, 0."
Salah says, "I'm trying to move to 4, 0."
    Tock
Sally asks, "Am I close?"
  Harry says to Sally, "You're cool."
Sue says, "I'm trying to move to 5, 0."
Salah says, "I'm trying to move to 4, 0."
Sara asks, "Am I close?"
  Harry says to Sara, "You're cool."
    Tick
Sue says, "I'm trying to move to 5, 0."
Salah says, "I'm trying to move to 4, 0."
    Tock
Sally says, "I took a step.  I'm now at 6, 0."
Sue says, "I took a step.  I'm now at 5, 0."
Salah says, "I took a step.  I'm now at 4, 0."
Sara says, "I took a step.  I'm now at 0, 6."
    Tick
    Tock
Sally asks, "Am I close?"
  Harry says to Sally, "You're cool."
Sue asks, "Am I close?"
  Harry says to Sue, "You're cool."
Salah asks, "Am I close?"
  Harry says to Salah, "You're warm."
Sara asks, "Am I close?"
  Harry says to Sara, "You're cool."
    Tick
    Tock
Sally says, "I took a step.  I'm now at 6, 1."
Sue says, "I'm trying to move to 6, 1."
Salah says, "I took a step.  I'm now at 5, 1."
Sara says, "I took a step.  I'm now at 1, 6."
    Tick
Sue says, "I'm trying to move to 6, 1."
Salah says, "I'm trying to move to 6, 1."
    Tock
Sally asks, "Am I close?"
  Harry says to Sally, "You're cool."
Sue says, "I'm trying to move to 6, 1."
Salah says, "I'm trying to move to 6, 1."
Sara asks, "Am I close?"
  Harry says to Sara, "You're cool."
    Tick
```

```
Sue says, "I'm trying to move to 6, 1."
Salah says, "I'm trying to move to 6, 1."
    Tock
Sally says, "I'm trying to move to 5, 1."
Sue says, "I'm trying to move to 6, 1."
Salah says, "I'm trying to move to 6, 1."
Sara says, "I took a step.  I'm now at 1, 5."
    Tick
Sally says, "I'm trying to move to 5, 1."
Sue says, "I'm trying to move to 6, 1."
Salah says, "I'm trying to move to 6, 1."
    Tock
Sally says, "I'm trying to move to 5, 1."
Sue says, "I'm trying to move to 6, 1."
Salah says, "I'm trying to move to 6, 1."
Sara asks, "Am I close?"
  Harry says to Sara, "You're warm."
    Tick
Sally says, "I'm trying to move to 5, 1."
Sue says, "I'm trying to move to 6, 1."
Salah says, "I'm trying to move to 6, 1."
    Tock
Sally says, "I'm trying to move to 5, 1."
Sue says, "I'm tired of waiting!"
Salah says, "I'm trying to move to 6, 1."
Sara says, "I took a step.  I'm now at 1, 4."
    Tick
Sally says, "I'm trying to move to 5, 1."
Sue says, "I give up!  I took 5 steps before quitting."
Salah says, "I'm tired of waiting!"
    Tock
Sally says, "I'm trying to move to 5, 1."
Salah says, "I give up!  I took 5 steps before quitting."
Sara asks, "Am I close?"
  Harry says to Sara, "You're hot."
    Tick
Sally says, "I'm trying to move to 5, 1."
```

Screenshot:



Marking notes:

- 1 mark, for adding the "I'm tired of waiting" feature. Note that there should be some diagnostic output on the console to indicate when a seeker is waiting.
- 1 mark, for adding the second command-line argument (1 to 10 seekers).
- 2 marks, for adding the "only one seeker per cell" feature.
- 2 marks, for adding a "clever seeker" who chooses a (nextX, nextY) destination whose Temperature is unknown).
- If the student doesn't submit a jarfile with source code, or if the jarfile doesn't compile and run with default arguments, award 0 marks for this question.