```java
package sample;

import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import java.util.ArrayList;
import java.util.Random;

/**
 * 9/9/19
 * Currently a bug is causing the program to not draw the miners
 *
 * the current progject is to implement working buildings and the miners is the first of said objects,
 * the next building to implement will be the conveyors then inserters then furnace, then assembler,
 *
 * next an integration of power for the power lines and the generator should be implented along with the pipe system
 */

import static java.lang.Math.*;

        public class Main {

            public static JFrame frame = new JFrame("Frog Version: July 31, 2019");


            public Main(){

                Container c = frame.getContentPane();
                c.setBackground(Color.red);

                frame.setBackground(Color.yellow);


                frame.setLocationRelativeTo(null);
                GameCanvas program = new GameCanvas();
                frame.add(program);
                frame.setVisible(true);

                //On Close of game window go back to the menu window
```

```
45                  frame.addWindowListener(new WindowAdapter() {
46              @Override
47              public void windowClosing(WindowEvent e) {
48                  System.out.println("Closing Window");
49
50                  FileReader ouputData = new FileReader("
   DataOutput.txt");
51
52                  for(String line: program.getDateOutput()){
53                      ouputData.addLine(line);
54                      ouputData.addLine("FUnnny");
55                  }
56
57                  Menu.makeVisible();
58              }
59          });
60      }
61  }
62      class GameCanvas extends JComponent {
63
64          private Assembler defaultAssembler;
65          private Conveyor defaultConveyor;
66          private EightBuilding defaultEightBuilding;
67          private ElectricPole defaultElectricPole;
68          private MinerBuilding defaultMinerBuilding;
69          private NinthBuilding defaultNinthBuilding;
70          private Pipe defaultPipe;
71          private SeventhBuilding defaultSeventhBuilding;
72          private SteamGenerator defaultSteamGenerator;
73          private Player defaultPlayer;
74
75          private SolidObject defaultSolidObject;
76          private MovingObject defaultMovingObject;
77          private Plane defaultPlane;
78          private Frog defaultFrog;
79
80          //default list of ores
81          private CopperOre defaultCopper;
82          private IronOre defaultIron;
83          private CoalOre defaultCoal;
84
85          //General World Settings
86          private boolean gamePaused = false; // this is a toggle
    for the 'p' button to pause all movement players and arrows at
    the time of creation but potentially enemies
87
88          private boolean graphicsOn = true;
89
90          private int initPopulationSize;//used as the beginning
   population of the frogs
```

```
 91
 92
 93
 94
 95          private int GameSpeed = 1;//each increase is in every
    loop is how many times it per count eg: if 2 then every 10
    millseconds all calculations are run twice
 96
 97          private boolean isDebug; //if true the score board
    function will display a lot of information
 98
 99          private int pelletCount = 1; // the number of pellets
    added per round
100
101          //framecount using maths can sorta be used to get
    seconds / minutes ect but can be out of sync due to program /
    hardware lag
102          private int framecount=1; //the total count of all
    frames for the duration of the program running
103
104          private int roundDuration= 2;//this is in minutes
105          private int roundCount = 0;//this is the current round
     count
106
107          private int tempRoundCount = 0;
108
109          private Random random = new Random(); // called in
    various places; mostly used to get a random nuber in a range
    using nextInt()
110
111          //use a variable size player list to allow for more
    players later on / to allow for some to die
112          private  ArrayList<Frog> frogList;
113
114          private ArrayList<Player> playerList;
115
116          //this is the list of the 'food' items
117          private ArrayList<Food> pelletList;
118
119          /**
120           * Dynamic arraylist of all the buildings that are
    currently built
121           */
122          private ArrayList<Building> buildingsList  = new
    ArrayList<>();
123
124          private ArrayList<MinerBuilding>
    minerBuildingArrayList = new ArrayList<>();
125
126          private ArrayList<Ore> oreList = new ArrayList<>();
```

```
127
128         private ArrayList<DroppedItem> droppedItemsList;
129
130         public ArrayList<String> getDateOutput() {
131             return dateOutput;
132         }
133
134         public ArrayList<String> dateOutput = new ArrayList<
    String>();
135
136
137         private ArrayList<BackgroundImage> backgroundImageList
     = new ArrayList<BackgroundImage>();
138
139         //IMAGES FILE PATHS
140
141         //The background images
142         private  String backgroundFilePath;
143
144         private BufferedImage BACKGROUNDIMAGE;
145
146         //THE ACTUAL IMAGE OBJECT
147
148         private BufferedImage currentBackground;
149
150         private Map gameMap;
151
152         private devTools developerTool = new devTools();
153
154         /**
155          * The mouse listener is used to activate various
    actions when the player / user should use the mouse
156          */
157         MouseListener ratListner = new MouseListener() {
158             @Override
159             public void mouseClicked(MouseEvent e) {
160
161                 Building output = playerList.get(0).
    calcMouseClick(e, gameMap);
162
163                 if(output!=null
164                         &&output.getR_Image()!=null
165                 ) {
166
167                     if (buildingsList != null) {//always
    safety check
168
169                         //only add the selected tile if the
    button click is the primary click
170                         if (e.getButton() == 1) {
```

```
171                             boolean isOverlapping = false;
172
173                             //Loop through each existeing
    building to ensure no overlap
174                             if(buildingsList!=null)for (
    Building a : buildingsList) {
175
176                                 //Create sam and BuildingA
    with the width and length not in pixels but in tiles
177                                 /**
178                                  * Can't use collision becuase
     the collision detected secretly as a 1 pixel additional
    distance which
179                                  * overlaps b/c the tiles are
    directly adjacent
180                                  */
181                                 if (output.isCollision(a,
    gameMap)) {
182                                     isOverlapping = true;
183                                 }
184                             }
185                             //only add the new building in the
     event that it doesn't overlap with any other buildings
186                             if (!isOverlapping) {
187
188                                 buildingsList.add(output);
189
190                                 MinerBuilding SecondaryOutput
     = playerList.get(0).calcMouseMinerClick(e, gameMap);
191
192                                 if(SecondaryOutput!=null)
    minerBuildingArrayList.add(SecondaryOutput);
193
194                             }
195                         }
196                     }
197                 }
198
199
200                 repaint();
201
202             }
203
204             /**
205              * ToDo
206              * use the functions mouse pressed and mouse
    released to store a starting tile and ending tile
207              * from these two tiles attempt to place from the
    top left the maximum of the player selected building
208              * such buildings list should then indidually be
```

```
208 compared for the saefty functions eg: collision of buildings
209              *
210              * This should allow the player to place a set of
    objects in a rectangular space
211              *
212              * adding a player alt button possibly shift might
     allow for additional safety
213              *
214              * eg: draggin and relaseing the mouse will create
     either along the columns or rows
215              * but if the shift key is currenlty pressed then
    the mouse should default as described above wherin it goes
    along both the columns and rows
216              *
217              * this may help faster building and more intutive
     design
218              *
219              * such design should attempt to on some level
    provide a visual output of the currenlty selected tiles first
    by showing
220              * the rectangle and then later by showing
    graphically but softened to show they haven't been placed
    images of the selected object as they will be placed
221              *
222              */
223             @Override
224             /**
225              * This function will activate when a mouse button
     is pressed
226              */
227             public void mousePressed(MouseEvent e) {
228
229
230                 Building output = playerList.get(0).
    calcMouseClick(e, gameMap);
231                 //otherwise if the button click was the 'left
    ' or alt click then remove any items at the selected position
232                 if (e.getButton() == 3) {
233                     //Stop stupid stuff and just set the widht
     and lenght to 1 pixel
234                     output.setObjHeight(1);
235                     output.setObjWidth(1);
236                     if(buildingsList!=null)for (int i = 0; i
     < buildingsList.size(); i++) {
237
238                         //remove any colliding objects from
    the building list that collide with the slected tile
239                         /**
240                          * Can't use collision becuase the
    collision detected secretly as a 1 pixel additional distance
```

```
240 which
241                          * overlaps b/c the tiles are directly
     adjacent
242                          */
243                         if (output.isCollision(buildingsList.
    get(i),gameMap)) {
244                             buildingsList.remove(i);
245
246                             //safety checks to stop index out
    of bounds errors
247                             if (i < 0) i = 0;
248                             if (i > buildingsList.size())
    break;
249                         }
250                     }
251
252                     if(minerBuildingArrayList!=null)for (int i
     = 0; i < minerBuildingArrayList.size(); i++) {
253
254                         //remove any colliding objects from
    the building list that collide with the slected tile
255                         /**
256                          * Can't use collision becuase the
    collision detected secretly as a 1
257                          * pixel additional distance which
258                          * overlaps b/c the tiles are directly
     adjacent
259                          */
260                         if (output.isCollision(
    minerBuildingArrayList.get(i),gameMap)) {
261                             minerBuildingArrayList.remove(i);
262
263                             //safety checks to stop index out
    of bounds errors
264                             if (i < 0) i = 0;
265                             if (i > minerBuildingArrayList.
    size()) break;
266                         }
267
268
269                     }
270                 }
271             }
272
273             @Override
274             /**
275              * Activates whenever a mouse button is released
276              */
277             public void mouseReleased(MouseEvent e) {
278
```

```
279                 Building output = playerList.get(0).
    calcMouseClick(e, gameMap);
280                 //otherwise if the button click was the 'left
    ' or alt click then remove any items at the selected position
281                 if (e.getButton() == 3) {
282                     //Stop stupid stuff and just set the widht
     and lenght to 1 pixel
283                     output.setObjHeight(1);
284                     output.setObjWidth(1);
285                     if(buildingsList!=null)for (int i = 0; i
     < buildingsList.size(); i++) {
286 
287                         //remove any colliding objects from
    the building list that collide with the slected tile
288                         /**
289                          * Can't use collision becuase the
    collision detected secretly as a 1 pixel additional distance
    which
290                          * overlaps b/c the tiles are directly
     adjacent
291                          */
292                         if (output.isCollision(buildingsList.
    get(i),gameMap)) {
293                             buildingsList.remove(i);
294 
295                             //safety checks to stop index out
    of bounds errors
296                             if (i < 0) i = 0;
297                             if (i > buildingsList.size())
    break;
298                         }
299                     }
300 
301                     if(minerBuildingArrayList!=null)for (int i
     = 0; i < minerBuildingArrayList.size(); i++) {
302 
303                         //remove any colliding objects from
    the building list that collide with the slected tile
304                         /**
305                          * Can't use collision becuase the
    collision detected secretly as a 1
306                          * pixel additional distance which
307                          * overlaps b/c the tiles are directly
     adjacent
308                          */
309                         if (output.isCollision(
    minerBuildingArrayList.get(i),gameMap)) {
310                             minerBuildingArrayList.remove(i);
311 
312                             //safety checks to stop index out
```

```
312 of bounds errors
313                             if (i < 0) i = 0;
314                             if (i > minerBuildingArrayList.
    size()) break;
315                         }
316
317
318                     }
319                 }
320
321
322                 output = playerList.get(0).calcMouseClick(e,
    gameMap);
323
324                 if(output!=null
325                         &&output.getR_Image()!=null
326                 ) {
327
328                     if (buildingsList != null) {//always
    safety check
329
330                         //only add the selected tile if the
    button click is the primary click
331                         if (e.getButton() == 1) {
332                             boolean isOverlapping = false;
333
334                             //Loop through each existeing
    building to ensure no overlap
335                             if(buildingsList!=null)for (
    Building a : buildingsList) {
336
337                                 //Create sam and BuildingA
    with the width and length not in pixels but in tiles
338                                 /**
339                                  * Can't use collision becuase
     the collision detected secretly as a 1 pixel additional
    distance which
340                                  * overlaps b/c the tiles are
    directly adjacent
341                                  */
342                                 if (output.isCollision(a,
    gameMap)) {
343                                     isOverlapping = true;
344                                 }
345                             }
346                             //only add the new building in the
     event that it doesn't overlap with any other buildings
347                             if (!isOverlapping) {
348
349                                 buildingsList.add(output);
```

```
350                                 }
351                             }
352                         }
353                     }
354
355
356                     repaint();
357                 }
358
359                 @Override
360                 /**
361                  * This function will trigger with the first position of the mouse as it enters the window
362                  */
363                 public void mouseEntered(MouseEvent e) {
364
365                 }
366
367                 @Override
368                 /**
369                  * This function will trigger with the last position of the mouse as it exits the window
370                  */
371                 public void mouseExited(MouseEvent e) {
372                 }
373             };
374
375             /**
376              * The InputTracker is used to track keyboard actions as both listed under the developer commands and the
377              * various commands of the players in the player lists
378              * ToDo
379              * I would like to see a comination of commands eg: shift + r to reverese the direction of rotation
380              *
381              */
382             KeyListener InputTracker = new KeyListener() {
383
384                 public void keyPressed(KeyEvent e) {
385                     calcPlayerInput(e);
386
387                     /**
388                      * This function is given e
389                      * @param e - a keyboard input
390                      * Then test the keyboard button against the dev buttons and activate various commands as needed
391                      * @param graphicsOn
392                      * @param gameMap
393                      * @param gamePaused
394                      * @param Keycmd_PauseGame
```

```
395                  * @param Keycmd_repopulateFood
396                  * @param Keycmd_ToggleGraphics
397                  * @param Keycmd_IncreaseSpeed
398                  * @param Keycmd_DecreaseSpeed
399                  */
400                 developerTool.calcCommands(e,graphicsOn,
    gamePaused,gameMap, playerList);
401
402
403                 int key = e.getKeyCode();
404             }
405             public void keyTyped(KeyEvent e){
406
407
408             }
409
410             public void keyReleased(KeyEvent e) {
411                 Player.calcPlayerReleasedInput(e,playerList);
412             }
413
414         };
415
416         private void InitializeDefaultValues() {
417             Color defaultC = new Color(50,50,50);
418
419             System.out.println("InitializeDefaultValues");
420
421             defaultAssembler = new Assembler(0,0,0,0,defaultC,
422                     gameMap, null,null,null,null, false);
423
424             defaultConveyor = new Conveyor(0,0,0,0,defaultC,
425                     gameMap, null,null,null,null, false);
426
427              defaultEightBuilding = new EightBuilding(0,0,0,0,
    defaultC,
428                      gameMap, null,null,null,null, false);
429
430               defaultElectricPole = new ElectricPole(0,0,0,0,
    defaultC,
431                       gameMap, null,null,null,null, false);
432
433              defaultMinerBuilding = new MinerBuilding(0,0,0,0,
    defaultC,
434                      gameMap, null,null,null,null, false);
435
436              defaultNinthBuilding = new NinthBuilding(0,0,0,0,
    defaultC,
437                      gameMap, null,null,null,null, false);
438
439              defaultPipe = new Pipe(0,0,0,0,defaultC,
```

```
440                         gameMap, null,null,null,null, false);
441
442                 defaultSeventhBuilding = new SeventhBuilding(0,0,
    0,0,defaultC,
443                         gameMap, null,null,null,null, false);
444
445                 defaultSteamGenerator = new SteamGenerator(0,0,0,
    0,defaultC,
446                         gameMap, null,null,null,null, false);
447
448                 defaultSolidObject = new SolidObject(0,0,0,0,
    defaultC);
449                 defaultMovingObject = new MovingObject(0,0,0,0,0,
    0);
450
451                 defaultPlayer = new Player(0,0,0,0,0,0,
452                         0,0,0,0,0,0,0,
453                         "",0,0,0,0,null);
454
455                 //default ORE initalizations prior to reading
    files
456                 defaultCoal = new CoalOre(0,0,0,0,defaultC,
457                         null, null,null,null);
458                 defaultIron = new IronOre(0,0,0,0,defaultC,
459                         null, null,null,null);
460                 defaultCopper = new CopperOre(0,0,0,0,defaultC,
461                         null, null,null,null);
462
463
464                 //THE BLUE PLAYER FILE PATH
465                // blueJetFilePath = "BlueJet.png";
466                 //blueJetFlipPath = "BlueJetFlipped.png";
467
468                 //The background image
469                 backgroundFilePath = "";
470
471                 //General World Settings
472                 gamePaused = false; // this is a toggle for the 'p
    ' button to pause all movement players and arrows at the time
    of creation but potentially enemies
473
474                 initPopulationSize = 1;
475
476                 //the following must have additional lines for
    additional built players
477
478
479
480                 isDebug = false;
481
```

```
482             //GROUND TROOP VALUES
483             //use a variable size player list to allow for
    more players later on / to allow for some to die
484             frogList = new ArrayList<>();
485
486             playerList = new ArrayList<>();
487
488             buildingsList = new ArrayList<>();
489
490             pelletList = new ArrayList<>();
491
492             droppedItemsList = new ArrayList<>();
493
494             graphicsOn = true;
495
496             int roundDuration= 2;//this is in minutes
497         }
498
499         /**
500          *
501          */
502         protected GameCanvas() {
503
504         gameMap = new Map(0,0,400,400,
505                 10000,10000, 40,40
506                 , new Color(150, 75, 0),
507                 new Color(160, 75, 0)
508         );
509
510         InitializeDefaultValues();
511
512         populateDefaultOres();
513         System.out.println("GameCanvas: "+(defaultCoal.
    getUp_Image()!=null));
514
515         //inialize the ore for the gamemap
516         for(Ore a: gameMap.populateOre(defaultCoal, 10, 10, 20
    , 20))oreList.add(a);
517
518         for(Ore a: gameMap.populateOre(defaultCopper, 100, 10
    , 20, 20))oreList.add(a);
519
520         for(Ore a: gameMap.populateOre(defaultIron, 10, 100,
    20, 20))oreList.add(a);
521
522         System.out.println("GameCanvas");
523
524         gamePaused = false;
525
526         populateDefaultVariables();
```

```
527
528         firstTimeinitialization();
529     }
530
531
532 private void overrideGameValues(String fileName) {
533
534     System.out.println("\noverrideGameValues\n");
535
536     FileReader file = new FileReader(fileName);
537
538     //Overrride all the player values
539     OverrideAllPlayerValues(playerList);
540
541     //Finally handle overriding the game cmd buttons
542     OverridingValuesClass.OverrideGameCmds(file, pelletCount,
    gamePaused, isDebug,
543             graphicsOn, roundDuration, initPopulationSize,
    developerTool.getKeycmd_IncreaseSpeed(),
544             developerTool.getKeycmd_DecreaseSpeed(),
    developerTool.getKeycmd_repopulateFood(),
545             developerTool.getKeycmd_ToggleGraphics(),
    developerTool.getKeycmd_StepRound(),
546             developerTool.getKeycmd_repopulateFood()
547             );
548 }
549
550
551         /**
552          * @param list This is at time of creation the player
    list which is referenced against internal values:
553          * KeyBoard Inputs: buttonUP, buttonDown, buttonLeft,
    buttonRight ,buttonFire, buttonAltFire
554          * Player Values: Height, Width, VSpeed (and sets the
    default), HSpeed (and sets the default), health, name (
    entirely for role play)
555          *  Player reference values: FIRECOOLDOWN,
    BOMBCOOLDOWN, DefaultProjectileHeight, DefaultProjectileWidth
556          *
557          *  and will safely loop through and override if any
    such values are found
558          *
559          *  This looping allows for easy changes to the
    numbers of players without having to add additional code but
560          *              means that to set a value it will look
    for buttonUp = 'Player_' + (the position, yes from 0)+'
    buttonUp'
561          *              so as a whole it might set the 3rd
    players buttonAltFire = 'Player_2_buttonAltFire'
562          */
```

```
563
564         private void OverrideAllPlayerValues(ArrayList<Player
    > list) {
565
566             System.out.println("OverrideAllPlayerValues");
567
568
569             FileReader file = new FileReader("Players Model\\
    Player_0\\playersettings.txt");
570             String temp = "";
571             String players_folder = "Players Model\\Player_";
572             String fileName = "playersettings.txt";
573             String type = "Player";
574
575
576
577             for (int position = 0; position< list.size();
    position++) {
578
579                 Player self = list.get(position);
580
581                 // file.setFileName(players_folder+position
    +"/"+fileName);
582                 // file.setFileName(fileName+position+"\\"+
    fileName);
583
584                 file.setFileName("Players Model\\Player_0\\
    playersettings.txt");
585
586                 file.setFileFolder("Players Model\\Player_0\\"
    );
587
588                 System.out.println("OverridingPlayer: "+file.
    getFileName());
589             }
590
591
592             basePopulatePlayers(1,playerList,"
    OverrideAllPlayerValues");
593         }
594
595
596
597
598         /**
599          * Given e handle any relevant action that should
    occur with the players
600          * @param e - Keyevent
601          *
602          * calls ArrayList PlayerList<Plane>
```

```
603          *
604          * This will move the plane on a speed fom the planes default speed value
605          *
606          */
607         private void calcPlayerInput(KeyEvent e){
608
609             int key = e.getKeyCode();
610
611
612             /**
613              * Loop through each plane in the arraylist and handle the relevant action if any match each individuals list of actions
614              */
615
616             if(playerList!=null)for(Player self: playerList) {
617
618                 if (self != null) {
619
620                     //calcplayer belt selection changes
621                     self.BeltItemKeyEvent(e);
622
623                     //UP Key
624                     if (self.getButtonUp() == key) {
625                         //handle moving the plane / player in the requested direction
626                         //Move the player up by negativify an absolute of the default value
627
628                         self.setObjVSpeed(
629                                 -abs(self.getDefaultVSpeed()
630                                 ));
631
632                     } else //DOWN Key
633                         if (self.getButtonDown() == key) {
634                             //handle moving the plane / player in the requested direction
635                             //Move the player up by absolute of the default value
636
637                             self.setObjVSpeed(
638                                     abs(self.getDefaultVSpeed
639                                     ));
640                         } else //LEFT Key
641                             if (self.getButtonLeft() == key) {
642                                 //handle moving the plane / player in the requested direction
643                                 //Move the player left by
```

```
643 negativify an absolute of the default value
644
645                                     self.setObjHSpeed(
646                                             -abs(self.
    getDefaultHSpeed()
647                                             ));
648                                 } else //RIGHT Key
649                                     if (self.getButtonRight() ==
    key) {
650                                         //handle moving the plane
     / player in the requested direction
651                                         //Move the player right an
     absolute of the default speed value
652
653                                         self.setObjHSpeed(
654                                                 abs(self.
    getDefaultHSpeed()
655                                                 ));
656                                     } else //FIRE Key
657                                         if (self.getButtonFire
    () == key) {
658                                             self.rotateRight();
659
660                                         } else //ALT FIRE Key
661                                             if (self.
    getButtonAltFire() == key) {
662
663                                             }
664                     self.calcMovement();
665                 }
666             }
667         }
668
669
670
671
672         private void intializeImages() {
673
674             BACKGROUNDIMAGE = null;
675             currentBackground = null;
676         }
677
678         private BufferedImage imageGetter(String filePathName
    ) {
679             try {
680
681                 return ImageIO.read(new File(filePathName));
682             } catch (IOException e) {
683
684
```

```
685                 System.out.println(e.toString());
686             }
687             return null;
688         }
689 
690         private void firstTimeinitialization() {
691 
692             System.out.println("firstTimeinitialization");
693 
694 
695             //use prebuilt values, make players and put them
    into the frogList arrayList
696 
697             String temp = "PlayerCount";
698             FileReader file = new FileReader("GameSettings.txt
    ");
699 
700             int value = 0;
701 
702             if (!file.findValue(temp).equals("")
703                     && file.convertStringToInt(file.findValue(
    temp)) != -1)
704                 value =
705                         file.  convertStringToInt(file.
    findValue(temp))
706                 ;
707 
708 
709             overrideGameValues("GameSettings.txt");
710 
711             intializeImages();
712 
713             //make sure that the window will actually listen
    for inputs
714             initListeners();
715 
716            Thread animationThread = new Thread(new Runnable
    () {
717                 public void run() {
718                     while (true) {
719                         repaint();
720                         try {
721                             Thread.sleep(10);
722                         } catch (Exception ex) {
723                         }
724                     }
725                 }
726             });
727             animationThread.start();
728         }
```

```
729
730         public void initListeners() {
731             this.addKeyListener(InputTracker);
732             this.addMouseListener(ratListner);
733
734             this.setFocusable(true);
735         }
736
737
738
739
740         public void paintComponent(Graphics g) {
741
742             int frameSeconds =0; //calculates the seconds per
    the frame count
743             int frameMinutes = 0; // calculates the minutes
    based on the seconds which come from the frame count
744
745             Graphics2D gg = (Graphics2D) g;
746
747             gameMap.setViewHeight(this.getHeight());
748             gameMap.setViewWidth(this.getWidth());
749
750             gameMap.updatePosition(playerList.get(0));
751
752             gameMap.drawCheckerboard(gg);
753
754             gameMap.drawBackgroundImages(gg,
    backgroundImageList);
755
756             //stop the program doing anything when the program
     is paused
757
758                 //loop the game per regular cycle timers the
    game speed which means that there is a functional fast forward
     button
759                 for(int i = 0;i<GameSpeed;i++) {
760
761                     //Only draw each object if the graphics
    are on and only calculate the movmenet if the game is not
    paused
762
763                     if(graphicsOn) {
764                         /**
765                          * The order that these following
    lines are very important
766                          * this is the order that things are
    drawn,
767                          * the last on on the list gets to
    draw over everyone else and thus will appear if overlapping
```

```
767 with another object
768                          */
769
770                         Ore.drawOre(gg,oreList,gameMap);
771
772                         Food.drawFood(gg, pelletList, gameMap
    );
773                         Building.drawBuildings(gg,
    buildingsList, gameMap);
774
775                         Frog.drawFrog(gg, frogList, gameMap, !
    gamePaused);
776
777                         Player.drawPlayers(gg, playerList,
    gameMap, !gamePaused);
778
779                         playerList.get(i).drawUI(gg, gameMap);
780
781                         if(!isDebug)developerTool.
    drawScorebaord(gg, framecount, frameSeconds, frameMinutes
782                         , roundCount, graphicsOn, gameMap,
    minerBuildingArrayList, buildingsList,
783                                 oreList, playerList, frogList
784                                 ,defaultMinerBuilding
785
786                         );
787                     }
788
789                     framecount++;
790
791                     frameSeconds = (framecount / 60); //
    calculates the seconds per the frame count
792                     frameMinutes = frameSeconds / 60; //
    calculates the minutes based on the seconds which come from
    the frame count
793
794                     //NEW ROUND
795                     /**
796                      * Given RoundDuration calculate when a
    new round occurs
797                      *
798                      * eg: round duration = 2 //it's in
    minutes
799                      * everytime the frame minutes is evenly
    divisble into it then start a roun
800                      */
801                     try {
802
803                         if (roundCount < frameMinutes /
    roundDuration
```

```
804                             //|| noMovesLeft
805                         ) {
806                             roundCount++;
807                             calcNewRound();
808                         }
809                     } catch (Exception e) {
810                     }
811 
812                     //this function along with subfunctions handles collisions between most objects
813                     calcCollisions();
814 
815                     repaint();
816     }
817 
818 
819             }
820 
821 
822 
823         public void calcNewRound(){
824      //this is run to populate a new round
825     }
826 
827 /**
828  * This function calculates and handles the
                    collisions for the following arrayLists:
829  *
830  * This function should be called before any objects are drwawn
831  * Explosions: explosionList
832 * Frog: frogList
833 * GroundFighters: groundFighters
834 * Projectiles: projectileList
835 */
836 protected void calcCollisions(){
837     //calcPlaneOnPlaneCollision(frogList, frogList);
838     if(pelletList!=null&& frogList!=null)
    calcPlaneOnFoodCollision(frogList,pelletList);
839     calcBuildingCollisions();
840 }
841 
842         protected void calcBuildingCollisions(){
843     if(buildingsList!=null&&droppedItemsList!=null){
844     //Compare each building against the list of moving objects
845     //this is going to typically be any dropped items (dropped items do include items on conveyors)
846     //additionally this will be the player too
847     for(Building building: buildingsList){
848 
```

```
849         //Calc collisions with dropped items
850         //this is primarly (entirely) used by the conveyor
    class
851         for(DroppedItem droppedItem: droppedItemsList){
852
853             if(building.isCollision(droppedItem,gameMap)){
854                 building.calcCollsion(droppedItem);
855             }
856         }
857
858         /**
859          * Calculate a collision of a building against the
    player
860          * When this heppens the player should bounce off the
    building unless it is a solid building
861          */
862         for(Player player: playerList){
863             if(building.isCollision(player,gameMap)){
864
865                 if(building.isSolid()) {
866                     //TOP
867                     if (player.isCollision(
868                             building.getPosX(), building.
    getPosY(), building.getObjWidth(), 1, gameMap
869                     )) {
870                         player.setPosY(
871                                 building.getPosY() -
872                                         player.getObjHeight()
873                         );
874                     } else if (
875                         //BOTTOM
876                             player.isCollision(
877                                     building.getPosX(),
878                                     (building.getPosY()+
    building.getObjWidth())-1,
879                                     building.getObjWidth(),
880                                     1, gameMap
881                             )
882                     ) {
883                         player.setPosY(
884                                 building.getPosY() + building.
    getObjHeight()
885
886                         );
887                     }
888                     //LEFT
889                     if (player.isCollision(
890                             building.getPosX(),
891                             building.getPosY(),
892                             1,
```

```
893                                 building.getObjHeight(), gameMap
894                         )) {
895                             player.setPosX(
896                                     building.getPosX()
897                                             -
898                                             player.getObjWidth()
899                             );
900                         } else if (
901                             //RIGHT
902                                 player.isCollision(
903                                         (building.getPosX()+
    building.getObjWidth())-1,
904                                         building.getPosY(),
905                                         1,
906                                         building.getObjHeight(),
    gameMap
907                                 )
908                         ) {
909
910                             player.setPosX(
911                                     building.getPosX() + building.
    getObjWidth()+1
912                             );
913
914                         }
915
916                     }
917
918                     else{
919                         building.calcCollsion(player);
920                     }
921
922                 }
923             }
924         }
925       }
926         calcMinerCollisions();
927 }
928
929 protected void calcMinerCollisions(){
930     if(minerBuildingArrayList!=null&&oreList!=null){
931         //Compare each building against the list of moving
    objects
932         //this is going to typically be any dropped items (
    dropped items do include items on conveyors)
933         //additionally this will be the player too
934         for(MinerBuilding miner: minerBuildingArrayList) {
935
936
937             //calculate collisions with ores
```

```
938             //used by the miners
939             //Compares the default Miner Building against the
    current iterated object to ensure only miner class objects are
     called
940             for (Ore a : oreList) {
941                 if (miner.isCollision(a,gameMap)
942                 ) {
943                     if (miner.calcMining()) {
944                         System.out.println("New Ore");
945                     }
946                 }
947             }
948         }
949     }
950 }
951
952
953 protected void calcPlaneOnFoodCollision(ArrayList<Frog> list,
    ArrayList<Food> gList) {
954
955     //loop through list safely
956     for (int a = 0; a < list.size(); a++) {
957
958         //loop through gList safely to allow for deletions in
    looping
959         for (int b = 0; b < gList.size(); b++) {
960
961 //use the isCollision function to find if these two objects in
     the two array list are colliding
962 if (
963                 list.get(a).isCollision(gList.get(b),gameMap
    )) {
964
965     //Handle when a collision is found with list.get(a) &
    gList.get(b)
966
967     list.get(a).increaseFoodSupply(1);//add a food counter to
    the frog
968
969     gList.remove(b);//remove the food pellet
970     b--; //offset for the removal
971
972     //finally safety check to break looping if the marker has
    exceeded the usable positions
973     if (b < 0 || b > gList.size()) break;
974 }
975 }
976 }}
977
978
```

```
 979         /**
 980          * This is typically called when the gamemap size
     needs to be updated
 981          * typically the gamemap tiles sizes are being
     updated
 982          */
 983         public void updateSets(){
 984             defaultMinerBuilding.setGameMap(gameMap);
 985 
 986               defaultAssembler.setGameMap(gameMap);
 987               defaultConveyor.setGameMap(gameMap);
 988               defaultEightBuilding.setGameMap(gameMap);
 989               defaultElectricPole.setGameMap(gameMap);
 990 
 991               defaultMinerBuilding.setGameMap(gameMap);
 992              defaultNinthBuilding.setGameMap(gameMap);
 993 
 994            defaultSeventhBuilding.setGameMap(gameMap);
 995             defaultSteamGenerator.setGameMap(gameMap);
 996 /*
 997             defaultFrog.setGameMap(gameMap);
 998             defaultPipe;.setGameMap(gameMap);
 999             defaultSolidObject.setGameMap(gameMap);
1000             defaultMovingObject.setGameMap(gameMap);
1001             defaultPlane.setGameMap(gameMap);
1002             defaultPlayer.setGameMap(gameMap);*/
1003 
1004             //default list of ores
1005               defaultCopper.setGameMap(gameMap);
1006               defaultIron.setGameMap(gameMap);
1007               defaultCoal.setGameMap(gameMap);
1008 
1009 
1010 
1011 
1012         }
1013 
1014 
1015 
1016         /**
1017          *
1018          * @return an arraylist containing the defaulted belt
      list of the player's belt
1019          */
1020         private ArrayList<BeltSlot> intializeBelt(boolean
     popdefaultvalues){
1021 
1022             System.out.println("intializeBelt");
1023 
1024         if(popdefaultvalues)populateDefaultVariables();
```

```
1025
1026            ArrayList<BeltSlot> list = new ArrayList<>();
1027
1028            list.add(new BeltSlot( 49, new Color(255, 15, 0),
     defaultAssembler,false));
1029
1030            list.add(new BeltSlot(50, new Color(57, 255, 0),
     defaultMinerBuilding,true));
1031
1032            list.add(new BeltSlot(51, new Color(255, 24, 217) ,
     defaultConveyor,false));
1033
1034            list.add(new BeltSlot(52, new Color(70, 255, 184),
     defaultSteamGenerator,false ));
1035
1036            list.add(new BeltSlot(53, new Color(109, 121, 255),
     defaultElectricPole,false));
1037
1038            list.add(new BeltSlot(54, new Color(37, 0, 255),
     defaultPipe,false));
1039
1040            list.add(new BeltSlot(55, new Color(179, 245, 255),
     defaultSeventhBuilding,false));
1041
1042            list.add(new BeltSlot(56, new Color(255, 255, 0),
     defaultEightBuilding,false));
1043
1044            list.add(new BeltSlot(57, new Color(255, 114, 44),
     defaultNinthBuilding,false));
1045
1046            populateDefaultVariables();
1047
1048            return list;
1049            }
1050
1051        /**
1052         KeyBoard Inputs: buttonUP, buttonDown, buttonLeft,
     buttonRight ,buttonFire, buttonAltFire
1053         Player Values: Height, Width, VSpeed (and sets the
     default), HSpeed (and sets the default), health, name (
     entirely for role play)
1054         Player reference values: FIRECOOLDOWN, BOMBCOOLDOWN,
     DefaultProjectileHeight, DefaultProjectileWidth
1055         * @param count - the amount of players to be added
1056         * @param list - the list of which to add them to
1057         */
1058        private void basePopulatePlayers(int count, ArrayList<
     Player> list, String calledby){
1059            System.out.println("basePopulatePlayers: "+ calledby
     );
```

```
1060
1061            System.out.println("Playervalue successful: "+ (
     defaultPlayer!=null));
1062
1063
1064            int x = gameMap.getMapWidth()/2;
1065            int y = gameMap.getMapHeight()/2;
1066
1067            String name = "PLAYER";
1068
1069            //Default KeyBoard values
1070            int defaultUp = 0;
1071            int defaultDown = 0;
1072            int defaultLeft =0;
1073            int defaultRight = 0;
1074            int defaultFire = 69;
1075            int defaultAltFire = 81;
1076
1077            //Default Plane values
1078            int width = 1;
1079            int height = 3;
1080            int VSpeed = 1;
1081            int HSpeed = 1;
1082             int health = 50;
1083
1084            //Reference Values
1085            int fireCool = 0;
1086            int bombCool = 0;
1087
1088            int projHeight = 100;
1089            int projWidth = 100;
1090
1091            ArrayList<BeltSlot> beltList = intializeBelt(true);
1092
1093                Player player = new Player(
1094                        x,y,width,height,HSpeed,VSpeed,
1095                        defaultUp,defaultDown,defaultLeft,
     defaultRight
1096                            ,defaultFire,defaultAltFire,health,
     name
1097                        ,fireCool,bombCool, projHeight,projWidth
1098                        ,beltList);
1099
1100                FileReader file = new FileReader("Players Model\\
     Player_0\\playersettings.txt");
1101                file.setFileFolder("Players Model\\Player_0\\");
1102            System.out.println("1309");
1103            OverridingValuesClass.OverridePlayer(defaultPlayer,
     file);
1104                System.out.println("\nDefaultOverClass: "+(
```

```
1104 defaultPlayer!=null)+"\n");
1105             System.out.println("Safely overriden player
     values: "+OverridingValuesClass.OverridePlayer(player,
     defaultPlayer));
1106 
1107             playerList.add(player);
1108 
1109 }
1110 
1111 
1112 
1113 /**
1114  * This function will call various sub functions to populate
     the default objects
1115  * the default objects are pre populated to store the various
      values of images and sizes, colours, ect
1116  * to prevent serious load issues on calling each objet as
     the function caries out its duties
1117  */
1118     private void populateDefaultVariables(){
1119     populateDefaultBuildings();
1120     populateDefaultOres();
1121     }
1122 
1123 
1124 /**
1125  *         private Assembler defaultAssembler;
1126  *         private Conveyor defaultConveyor;
1127  *         private EightBuilding defaultEightBuilding;
1128  *         private ElectricPole defaultElectricPole;
1129  *         private MinerBuilding defaultMinerBuilding;
1130  *         private NinthBuilding defaultNinthBuilding;
1131  *         private Pipe defaultPipe;
1132  *         private SeventhBuilding defaultSeventhBuilding;
1133  *         private SteamGenerator defaultSteamGenerator;
1134  *
1135  *         private SolidObject defaultSolidObject;
1136  *         private MovingObject defaultMovingObject;
1137  *
1138  *         private Plane defaultPlane;
1139  *         private Frog defaultFrog;
1140  */
1141 protected void populateDefaultBuildings(){
1142     System.out.println("populateDefaultBuildings");
1143 
1144     FileReader file = new FileReader("");
1145 
1146     String fileFolder = "";
1147 
1148     String fileName = "buildingSettings.txt";
```

```
1149
1150        String fileType = "";
1151
1152        //Buildings
1153        fileFolder="Buildings\\";
1154
1155        //Assembler
1156        fileType = "Assembler\\";
1157
1158        file = new FileReader("Buildings\\Assembler\\
        buildingSettings.txt");
1159        file.setFileFolder("Buildings\\Assembler\\");
1160        OverridingValuesClass.OverrideAssembler(defaultAssembler,
        file);
1161        defaultAssembler.setGameMap(gameMap);
1162
1163        //Conveyor
1164        fileType = "Conveyor\\";
1165
1166        System.out.println(fileFolder+fileType+fileName);
1167        file.setFileFolder(fileFolder+fileType);
1168        file.setFileName(fileFolder+fileType+fileName);
1169        OverridingValuesClass.OverrideConveyor(defaultConveyor,
        file);
1170        defaultConveyor.setGameMap(gameMap);
1171
1172        //defaultElectricPole
1173        fileType = "ElectricPole\\";
1174
1175        file.setFileName(fileFolder+fileType+fileName);
1176        file.setFileFolder(fileFolder+fileType);
1177        OverridingValuesClass.OVerrideElectricPole(
        defaultElectricPole,file);
1178        defaultElectricPole.setGameMap(gameMap);
1179
1180        //defaultMinerBuilding
1181        fileType = "MinerBuilding\\";
1182
1183        file.setFileName(fileFolder+fileType+fileName);
1184        file.setFileFolder(fileFolder+fileType);
1185        OverridingValuesClass.OverrideMinerBuilding(
        defaultMinerBuilding,file);
1186        defaultMinerBuilding.setGameMap(gameMap);
1187
1188        System.out.println("MinerBuilding Class: "+
        defaultMinerBuilding.getClass());
1189
1190        //defaultPipe
1191        fileType = "Pipe\\";
1192
```

```
1193        file.setFileName(fileFolder+fileType+fileName);
1194        file.setFileFolder(fileFolder+fileType);
1195        OverridingValuesClass.OverridePipe(defaultPipe,file);
1196        defaultPipe.setGameMap(gameMap);
1197
1198        //defaultSteamGenerator
1199        fileType = "SteamGenerator\\";
1200
1201        file.setFileName(fileFolder+fileType+fileName);
1202        file.setFileFolder(fileFolder+fileType);
1203        OverridingValuesClass.OVerrideSteamGenerator(
    defaultSteamGenerator,file);
1204        defaultSteamGenerator.setGameMap(gameMap);
1205
1206
1207        //defaultSeventhBuilding
1208        fileType = "SeventhBuilding\\";
1209
1210        file.setFileName(fileFolder+fileType+fileName);
1211        file.setFileFolder(fileFolder+fileType);
1212        OverridingValuesClass.OverrideSeventBuilding(
    defaultSeventhBuilding,file);
1213        defaultSeventhBuilding.setGameMap(gameMap);
1214
1215        //EightBuilding
1216        fileType = "EightBuilding\\";
1217
1218        System.out.println(fileFolder+fileType+fileName);
1219        file.setFileFolder(fileFolder+fileType);
1220        file.setFileName(fileFolder+fileType+fileName);
1221        OverridingValuesClass.OverrideEightBuilding(
    defaultEightBuilding,file);
1222        defaultEightBuilding.setGameMap(gameMap);
1223
1224        //defaultNinthBuilding
1225        fileType = "NinthBuilding\\";
1226
1227        file.setFileName(fileFolder+fileType+fileName);
1228        file.setFileFolder(fileFolder+fileType);
1229        OverridingValuesClass.OverrideNinthBuilding(
    defaultNinthBuilding,file);
1230        defaultNinthBuilding.setGameMap(gameMap);
1231
1232    }
1233
1234    /**
1235     * GameMap MUST be initalized before this is called
1236     * && the default Ores MUST BE inialized
1237     */
1238    private void populateDefaultOres() {
```

```
1239      System.out.println("populateDefaultOres");
1240
1241      FileReader Ofile = null;
1242      /*
1243      String fileType = "";
1244      String fileFolder = "ore\\";
1245      String fileName = "OreSettings.txt";
1246      */
1247      //COAL
1248      Ofile = new FileReader("ore\\coal\\OreSettings.txt");
1249      Ofile.setFileFolder("ore\\coal\\");
1250      OverridingValuesClass.OverrideSolidObject(defaultCoal,
     Ofile);
1251      defaultCoal.setGameMap(gameMap);
1252
1253      //COPPER
1254      Ofile = new FileReader("ore\\copper\\OreSettings.txt");
1255      Ofile.setFileFolder("ore\\copper\\");
1256      OverridingValuesClass.OverrideSolidObject(defaultCopper,
     Ofile);
1257      defaultCopper.setGameMap(gameMap);
1258
1259      //IRON
1260      Ofile = new FileReader("ore\\iron\\OreSettings.txt");
1261      Ofile.setFileFolder("ore\\iron\\");
1262      OverridingValuesClass.OverrideSolidObject(defaultIron,
     Ofile);
1263      defaultIron.setGameMap(gameMap);
1264
1265  }
1266
1267
1268          }
```