

```
1 import java.util.*;
2 /*****
3  * This class puts together specific values
4  * to build the game.
5  *
6  * @author Ben Parsell
7  * @version 1.0.0 (11/20/2015)
8  *****/
9 public class Game
10 {
11     /** ArrayList for player inventory */
12     private ArrayList<Item> playerInventory;
13
14     /** Player's last location */
15     private Room lastLocation;
16
17     /** Player's current location */
18     private Room currentLocation;
19
20     /** The Game's current message */
21     private String currentMessage;
22
23     /** Did you win? */
24     private boolean win;
25
26     /** Did you lose? */
27     private boolean lose;
28
29     /** Is it time to search a body? */
30     private boolean timeToSearch;
31
32     /** Gun ammo */
33     private int ammoCount;
34
35     /** Item variables */
36     private Item M1_Garand;
37     private Item grenade;
38     private Item helmet;
39     private Item pills;
40     private Item beer;
41     private Item ammo;
42     private Item apple;
43     private Item MRE;
44     private Item germanSoldier;
45     private Item mine;
46     private Item germanGeneral;
47     private Item tank;
48     private Item radio;
49 }
```

```
50     /** Room Variables **/  
51     private Room plane;  
52     private Room secondStoryFarmHouse;  
53     private Room firstStoryFarmHouse;  
54     private Room backyard;  
55     private Room openField;  
56     private Room cityEntrance;  
57     private Room drugStore;  
58     private Room apartments;  
59     private Room tavern;  
60     private Room germanCamp;  
61     private Room generalsTent;  
62     private Room hillTop;  
63  
64     /*****  
65     * Default Constructor  
66     *****/  
67     public Game() {  
68         // Player starts with gun  
69         playerInventory = new ArrayList<Item>();  
70  
71         // Creates rooms  
72         createRooms();  
73  
74         // Instantiate lastLocation  
75         lastLocation = null;  
76  
77         // Instantiate currentLocation as plane  
78         currentLocation = plane;  
79  
80         ammoCount = 4;  
81         // FIX ME (used for later)  
82         win = false;  
83         lose = false;  
84  
85     }  
86  
87  
88     /*****  
89     * Method to retrieve currentLocation  
90     *  
91     * @return currentLocation player's location  
92     *****/  
93     public Room getCurrentLocation() {  
94         return currentLocation;  
95     }  
96  
97  
98     /*****
```

```
99      * Method to set the initial message for player
100      *****/
101      public void setWelcomeMessage() {
102          currentMessage = "Welcome, soldier! You are a WWII 101st Airborne
paratrooper. You are dropping over northern France \n to help in the D-D
ay operation. You've noticed that the drop planes have overshot their des
tination though... \n";
103      }
104
105
106      /*****
107      * Method to check playerInventory for a specific
108      * item
109      *
110      * @return sub temp variable for item name
111      *****/
112      private Item checkForItem(String name) {
113          Item sub = null;
114          for(Item itm: playerInventory){
115              if(itm.getName().equals(name)){
116                  sub = itm;
117              }
118          }
119          return sub;
120      }
121
122
123      /*****
124      * Method to update currentMessage with list of
125      * player inventory items
126      *****/
127      public void list(){
128          currentMessage = "";
129          for(Item itm: playerInventory){
130              currentMessage = currentMessage + itm.getName() + " " + "\n";
131          }
132      }
133
134
135      /*****
136      * Method to create rooms and items
137      *****/
138      private void createRooms() {
139          // Create Items
140          M1_Garand = new Item("M1 Garand", "a beat up semi-auto rifle that
holds 8 rounds", 5, false);
141          grenade = new Item("Grenade", "an unused pineapple-style hand gre
nade", 1, false);
142          beer = new Item("Beer", "a bottle of warm beer", 3, true);
```

```

143     pills = new Item("Pills", "a bottle of pills with the prescriptio
n label ripped off", 1, true);
144     ammo = new Item("M1 Garand Ammo", "an 8 round ammo clip for the M
1 Garand Rifle, your gun is reloaded.", 1, false);
145     MRE = new Item("MRE", "a gross, meal that consists of canned corn
, cold pulled-pork, and candy", 1, true);
146     apple = new Item("Apple", "a fresh, Macintosh apple", 1, true);
147     germanSoldier = new Item("German Soldier", "a combat-ready soldie
r trying to kill you.", 200, false);
148     mine = new Item("Mine", "a primed, exlposive device ready to go o
ff if stepped on.", 250, false);
149     germanGeneral = new Item("German General", "a german general shin
ing with medals of his acomplishments", 210, false);
150     tank = new Item("Panzer Tank", "a Panzer Tiger tank loaded and re
ady to fire.", 3340, false);
151     radio = new Item("Radio", "a radio to call for reinforcements", 2
0, false);
152
153     // Create Rooms
154     plane = new Room("the airplane. It is taking heavy AA fire, you p
arachute out, and have to select a location to land", M1_Garand);
155     secondStoryFarmHouse = new Room("a destoryed-interior upstairs to
a house. There is a dead \nAmerican paratrooper on the floor", germanSol
dier);
156     firstStoryFarmHouse = new Room("You are in the main living room o
f the farmhouse. There are exits at the front and back-door,\n everything
else is blocked off.", germanSoldier );
157     backyard = new Room("A fenced-in backyard with a horse stable.",
germanSoldier);
158     openField = new Room("An open field filled with mines. They can s
afely be blown up...", mine);
159     cityEntrance = new Room("The gate to Main Street.", null);
160     drugStore = new Room("a traditional 1940s drug store with a soda
bar.", pills);
161     apartments = new Room("beat up apartments with bullet holes in th
e walls.", germanSoldier);
162     tavern = new Room("a French-style tavern. It's very dark in here.
..", germanSoldier);
163     germanCamp = new Room("the hefty german camp.", tank);
164     generalsTent = new Room("a shaggy military tent that has a radio
in it.", germanGeneral);
165     hillTop = new Room("a hilltop that has radio reception.", null);
166
167     // adding neighbors
168     plane.addNeighbor("north", secondStoryFarmHouse);
169     plane.addNeighbor("south", openField);
170
171     // Farm House 2nd Story
172     secondStoryFarmHouse.addNeighbor("downstairs" , firstStoryFarmHou

```

```

172 se);
173
174     // Farm House 1st Story
175     firstStoryFarmHouse.addNeighbor("north", backyard);
176     firstStoryFarmHouse.addNeighbor("upstairs", secondStoryFarmHouse)
177 ;
178     firstStoryFarmHouse.addNeighbor("south", cityEntrance);
179
180     // Open Field
181     openField.addNeighbor("west", cityEntrance);
182
183     // backyard
184     backyard.addNeighbor("south", firstStoryFarmHouse);
185
186     // City / Gate Entrance
187     cityEntrance.addNeighbor("east", openField);
188     cityEntrance.addNeighbor("south", drugStore);
189     cityEntrance.addNeighbor("north", firstStoryFarmHouse);
190
191     // Drugstore
192     drugStore.addNeighbor("north", cityEntrance);
193     drugStore.addNeighbor("east", tavern);
194     drugStore.addNeighbor("upstairs", apartments);
195
196     // Apartments
197     apartments.addNeighbor("downstairs", drugStore);
198
199     // Tavern
200     tavern.addNeighbor("west", drugStore);
201     tavern.addNeighbor("south", germanCamp);
202
203     // German Camp (Boss)
204     germanCamp.addNeighbor("north", tavern);
205     germanCamp.addNeighbor("east", generalsTent);
206
207     generalsTent.addNeighbor("north", hillTop);
208 }
209
210
211 /*****
212  * Method to determine when the game is over
213  *****/
214 public boolean gameOver() {
215     // Failed to kill upstairs enemy
216     if((currentLocation == firstStoryFarmHouse) && lastLocation == se
217 condStoryFarmHouse && lastLocation.getItem() == germanSoldier) {
218         currentMessage = "You failed to kill the enemy, and he shot y
219 ou in the chest dropping you dead.";

```

```
218         lose = true;
219         return true;
220     }
221
222     // Failed to kill mainfloor enemy
223     if((currentLocation == backyard) && (lastLocation == firstStoryFarmHouse) && (lastLocation.getItem() == germanSoldier)) {
224         currentMessage = "You failed to kill the enemy, and he shot you in the chest dropping you dead.";
225         lose = true;
226         return true;
227     }
228
229     // Failed to blow up mine
230     if((currentLocation == cityEntrance) && (lastLocation.getItem() == mine) && lastLocation == openField) {
231         currentMessage = "You failed to destroy the mine, and it threw your body into the air, shredding your limbs.";
232         lose = true;
233         return true;
234     }
235
236     // Failed to kill
237     if((currentLocation == backyard) && (lastLocation.getItem() == mine) && lastLocation == openField) {
238         currentMessage = "You failed to destroy the mine, and it threw your body into the air, shredding your limbs.";
239         lose = true;
240         return true;
241     }
242
243     // Failed to kill apartments enemy
244     if(currentLocation == drugStore && lastLocation == apartments && lastLocation.getItem() == germanSoldier) {
245         currentMessage = "You failed to kill the enemy, and he shot you in the chest dropping you dead.";
246         lose = true;
247         return true;
248     }
249
250     // Failed to kill first floor enemy
251     if((currentLocation == cityEntrance) && (lastLocation == firstStoryFarmHouse) && (lastLocation.getItem() == germanSoldier)) {
252         currentMessage = "You failed to kill the enemy, and he shot you in the chest dropping you dead.";
253         lose = true;
254         return true;
255     }
256
```

```
257         // Failed to kill backyard enemy
258         if(currentLocation == firstStoryFarmHouse && lastLocation == back
yard && lastLocation.getItem() == germanSoldier) {
259             currentMessage = "You failed to kill the enemy, and he shot y
ou in the chest dropping you dead.";
260             lose = true;
261             return true;
262         }
263
264         // Failed to kill tavern enemy
265         if(currentLocation == germanCamp && lastLocation == tavern && las
tLocation.getItem() == germanSoldier) {
266             currentMessage = "You failed to kill the enemy, and he shot y
ou in the chest dropping you dead.";
267             lose = true;
268             return true;
269         }
270
271         // Failed to kill tavern enemy
272         if(currentLocation == drugStore && lastLocation == tavern && last
Location.getItem() == germanSoldier) {
273             currentMessage = "You failed to kill the enemy, and he shot y
ou in the chest dropping you dead.";
274             lose = true;
275             return true;
276         }
277
278         // Ate posion pills and died
279         if(currentMessage.equals("You have eaten pills.\n")) {
280             currentMessage = "You thought they were tasty... but they poi
sioned you! Game over!";
281             lose = true;
282             return true;
283         }
284
285         // This is to remove ammo from inventory. Has no impact on gameOv
er, but it would
286         // have been unecessary to add a entire new method for just this
.
287         if(lastLocation == backyard && checkForItem("M1 Garand Ammo") !=
null) {
288             playerInventory.remove(ammo);
289             return false;
290         }
291
292
293         // Successfully beat game
294         if(currentLocation == hillTop && checkForItem("Radio") == radio &
& currentLocation.getItem() == null) {
```

```
295         currentMessage = "You called for reinforcements and successfu
lly helped with the D-Day operation!";
296         win = true;
297         return true;
298     }
299     // Otherwise, game continues
300     else {
301         return false;
302     }
303 }
304
305
306 /*****
307  * Method to move the player between rooms
308  *****/
309 public void move(String direction) {
310     // Create new room as you move
311     Room next = currentLocation.getNeighbor(direction);
312
313     // Error catch and actual movement
314     if( next == null) {
315         currentMessage = "You can't go that way" + "\n";
316     } else {
317         lastLocation = currentLocation;
318         currentLocation = next;
319         currentMessage = currentLocation.getLongDescription();
320     }
321 }
322
323
324 /*****
325  * Method to search body/continue on with game
326  *****/
327 public void continueGame() {
328     // Searches dead enemy on secondFloor
329     if(currentLocation == secondStoryFarmHouse) {
330         currentLocation.removeItem();
331         currentLocation.addItem(grenade);
332         currentMessage = "You search the enemy body and see something
. Pickup " + currentLocation.getItem().getName() + "?\n";
333         timeToSearch = false;
334     }
335
336     // Searches dead enemy in living room
337     if(currentLocation == backyard) {
338         currentLocation.removeItem();
339         currentLocation.addItem(ammo);
340         currentMessage = "You search the enemy body and see something
. Pickup " + currentLocation.getItem().getName() + "?\n";
```



```
341         ammoCount = 8;
342         timeToSearch = false;
343     }
344
345     // Searches dead enemy in backyard
346     if(currentLocation == firstStoryFarmHouse) {
347         currentLocation.removeItem();
348         currentLocation.addItem(apple);
349         currentMessage = "You search the enemy body and see something
. Pickup " + currentLocation.getItem().getName() + "?\n";
350         timeToSearch = false;
351     }
352
353     // Searches minefield
354     if(currentLocation == openField) {
355         currentLocation.removeItem();
356         currentLocation.addItem(apple);
357         currentMessage = "You search the now safe minefield and see s
omething. Pickup " + currentLocation.getItem().getName() + "?\n";
358         timeToSearch = false;
359     }
360
361     // Searches Tavern
362     if(currentLocation == tavern) {
363         currentLocation.removeItem();
364         currentLocation.addItem(beer);
365         currentMessage = "You search the enemy body and see something
. Pickup " + currentLocation.getItem().getName() + "?\n";
366         timeToSearch = false;
367     }
368
369     // Removes german from apartments
370     if(currentLocation == apartments) {
371         currentLocation.removeItem();
372         currentMessage = "You may move on.\n";
373         timeToSearch = false;
374     }
375
376     // Removes tank from camp
377     if(currentLocation == germanCamp) {
378         currentLocation.removeItem();
379         currentMessage = "You may now move on. \n ";
380         timeToSearch = false;
381     }
382
383     // Searches dead general
384     if(currentLocation == generalsTent) {
385         currentLocation.removeItem();
386         currentLocation.addItem(radio);
```

```
387         currentMessage = "You search the enemy body and see something
. Pickup " + currentLocation.getItem().getName() + "?\n";
388         timeToSearch = false;
389     }
390 }
391
392
393 /*****
394  * Method to fire your weapon
395  *****/
396 public void fire() {
397     // Check for rifle
398     if(checkForItem("M1 Garand") == null) {
399         currentMessage = "You do not have a rifle.\n";
400     }
401
402     // Check for ammo in rifle
403     else if(ammoCount < 1) {
404         currentMessage = "You are out of ammo.\n";
405
406         // Check for the general
407         }else if(currentLocation.getItem() == germanGeneral && checkForItem("M1 Garand") == M1_Garand && ammoCount > 0) {
408             currentMessage = "You killed the General! \n";
409             timeToSearch = true;
410             ammoCount -= 1;
411         }
412
413         // Check for a german soldier
414         else if(currentLocation.getItem() == germanSoldier && checkForItem("M1 Garand") == M1_Garand && ammoCount > 0) {
415             currentMessage = "You killed the enemy. CLICK CONTINUE \n";
416             timeToSearch = true;
417             ammoCount -= 1;
418
419             // Check for the mine
420             } else if(currentLocation.getItem() == mine && checkForItem("M1 Garand") == M1_Garand && ammoCount > 0) {
421                 currentMessage = "You shoot the mine, and a huge explosion of
dirt fills the sky. You may now move forward. CLICK CONTINUE\n";
422                 timeToSearch = true;
423                 ammoCount -= 1;
424
425                 // Check for the tank
426                 } else if(currentLocation.getItem() == tank && checkForItem("Grenade") == grenade) {
427                     currentMessage = "You ran up to the tank, jumped on it, and
threw the grenade into the hatch killing the crew. CLICK CONTINUE\n";
428                     playerInventory.remove("Grenade");
```

```
429         timeToSearch = true;
430     }
431
432     // Otherwise, you cannot fire now
433     else {
434         currentMessage = "You don't need to fire now, there are no en
emies.\n";
435     }
436 }
437
438
439 /*****
440  * Custom method to check if it is time to search
441  * an enemy body (works with buttons)
442  *****/
443 public boolean isTimeToSearch() {
444     // Check to see if it is time to search
445     if(timeToSearch) {
446         return true;
447     }
448     return false;
449 }
450
451
452 /*****
453  * Method to move the player between rooms
454  *****/
455 public void look() {
456     // Will display message to GUI later
457     currentMessage = currentLocation.getLongDescription() + "\n";
458 }
459
460
461 /*****
462  * Method to retrieve current message
463  *
464  * @return currentMessage the message for the game
465  *****/
466 public String getMessage() {
467     return currentMessage;
468 }
469
470
471 /*****
472  * Method to display a help message to the user
473  *****/
474 public void help() {
475     currentMessage = "You are an American paratropper behind enemy li
nes. You need to eliminate the German general before you can\n safely rec
```

```
475 connect with your men. Look for the German camp... \n";
476     }
477
478
479     /*****
480     * Method to pickup an item if there is one in
481     * the currentRoom
482     *****/
483     public void pickup(String item) {
484         // Check if room has an item at all
485         if(currentLocation.hasItem() == false){
486             currentMessage = "There is no item in the room to take" + "\n";
487         }
488
489         // Check if the item is too heavy to lift
490         else if(currentLocation.getItem().getWeight() > 100){
491             currentMessage = "That item is too heavy to pickup" + "\n";
492         }
493
494         // Otherwise, pickup the item
495         else if(item.equals(currentLocation.getItem().getName())){
496             playerInventory.add(currentLocation.getItem());
497             currentMessage = "You have picked up " + currentLocation.getItem().getName() + "\n";
498             currentLocation.removeItem();
499         }
500     }
501
502
503     /*****
504     * Method to drop an item from player inventory
505     *****/
506     public void drop(String item) {
507         Item roomItem = null;
508         boolean haveItem = false;
509
510         // Check to see if item already has an item
511         if(currentLocation.hasItem() == true) {
512             currentMessage = "Room has an item already.";
513         }
514
515         // Check to make sure inventory has an item
516         if(playerInventory.size() == 0) {
517             currentMessage = "You have nothing to drop.";
518         }
519
520         // Otherwise, run through inventory & drop
521         else {
```

```
522
523     // Cycles through inventory
524     for(Item i : playerInventory) {
525
526         // Check to make sure names match
527         if(i.getName().equals(item)) {
528             roomItem = i;
529             haveItem = true;
530             currentMessage = "You dropped an item.";
531         }
532     }
533
534     // Remove item and add it to room
535     playerInventory.remove(roomItem);
536     currentLocation.addItem(roomItem);
537
538     // Make sure you have an item
539     if(haveItem == false) {
540         currentMessage = "You aren't holding an item.";
541     }
542 }
543
544
545
546 /*****
547  * Eat an item if it's edible
548  *****/
549 public void eat(String item) {
550     Item canEat = null;
551
552     // Find item in player inventory
553     for(Item itm: playerInventory){
554         // Check for matches and set temp variable equal
555         if(itm.getName().equals(item)){
556             canEat = itm;
557         }
558     }
559
560
561     // Check to see if the item is in the inventory
562     if (canEat == null){
563         currentMessage = "You don't have that item"+ "\n";
564     }
565
566     // If it is, continue
567     else{
568
569         // Check to see if item is edible
570         if(canEat.isEdible() == true && item.equals("Pills")) {
```

```
571         currentMessage = "You have eaten pills.\n";
572         playerInventory.remove(canEat);
573     }
574     if(canEat.isEdible() == true && !item.equals("Pills")){
575         currentMessage = "Yum that was a tasty " + canEat.getName
576         ( ) + "!" + "\n";
577         playerInventory.remove(canEat);
578     }
579     // If it is not, update message
580     else if(canEat.isEdible() == false){
581         currentMessage = canEat.getName() + " is not edible." + "
582         \n";
583     }
584 }
585 }
586 }
```