# Decorator Simulator

## Backstory

In this game you play the role of a guy. You and your girlfriend have just bought an apartment and you are responsible for the decoration of the main room. Every first day of the month you are free to work on this. That day you can buy new stuff, or move items around in the room, anyway you like. At the end of that day, your girlfriend will come home from work to find out what you have done.

Now, being a guy, you tend to have simple goals. Your goal in the game is to have as many cool guy items in the room as possible. If it was up to you, the room would be filled with a pinball machine, minibar, stereo system, Xbox….Obviously, there's a catch…. You got to keep your girlfriend happy. Or at least happy enough so she won't leave you (then it's game over).

So, how do you do that ? Well, you could buy stuff she likes. That's pretty obvious, that will certainly uplift her mood. Of course, these items are usually items you don't like, so this will cost you points.
A second way has to do with the colours and the placement of the objects in the room. Your girlfriend is heavy into colours. She is very precise about it too. To that extent, that she has a version of the room in her head where colours are placed at specific positions.(= the ideal room). The more you can match the actual room to this ideal room, the more happy your girlfriend will be. So for example, a pinball machine is an item that your girlfriend does not like at all. But if you buy it in the right colour and put it in the right spot, the impact on your girlfriend's mood will be limited. That's the strategy to follow in other words. You can get away with murder if you manage this properly. You should use the comments of your girlfriend on the items to figure out the layout of this ideal room.

Also note that things you do (or don't do) in the game, can cause a one-time impact on the mood of your girlfriend and on your score (these are called events). These events can also just pop up randomly. You will have to play the game to see. Already one tip : you don't like to exercise, so moving heavy stuff around will cost you some points.

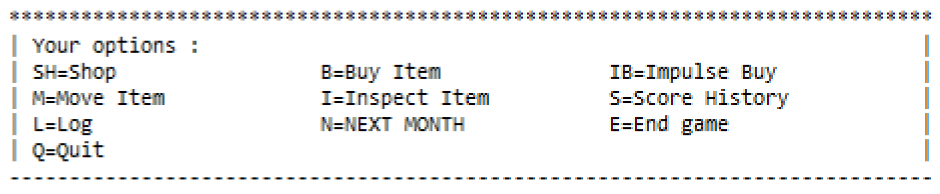
Every month your budget will increase. Beware, this is also true of your girlfriend's budget, and she might want to start spending this budget on stuff she likes..

If at any time you are pleased with the room and your score, you can end the game. Your score will be fixed.

So to summarize : get as many guy items in the room without causing your girlfriend to have a mental breakdown and leave you.

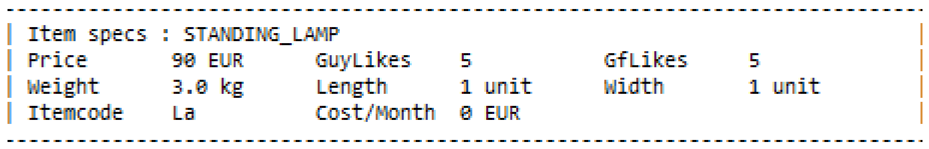
Have fun

## The options



```
*****************************************************************************
| Your options :                                                            |
| SH=Shop                 B=Buy Item              IB=Impulse Buy            |
| M=Move Item             I=Inspect Item          S=Score History           |
| L=Log                   N=NEXT MONTH            E=End game                |
| Q=Quit                                                                    |
-----------------------------------------------------------------------------
```

- **Shop**
  A complete overview of all the items you can buy



```
--------------------------------------------------------------------------------
| Item specs : STANDING_LAMP                                                   |
| Price       90 EUR      GuyLikes    5           GfLikes     5                |
| Weight      3.0 kg      Length      1 unit      Width       1 unit           |
| Itemcode    La          Cost/Month  0 EUR                                    |
--------------------------------------------------------------------------------
```

-> the more GuyLikes the items have, the more points you score
-> every item has a monthly maintenance cost
-> (width and length are not actively used in the current gameplay)

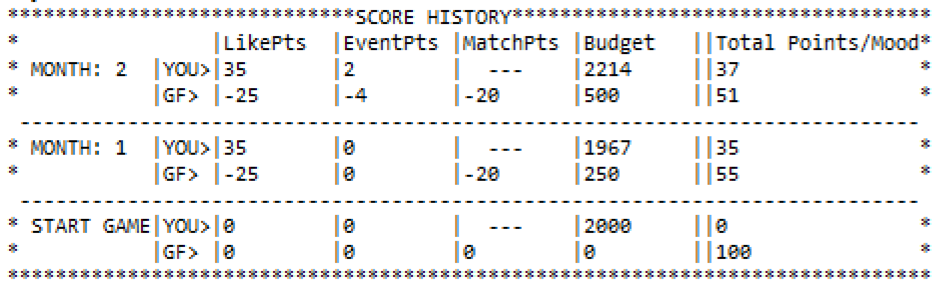
- **Buy**
  Buy an item and put it in the room.
  Be carefull, you will not be charged immediately (only after pressing NEXT MONTH)
  You can only buy a certain item once

- **Impulse Buy**
  buy a random item you like and put it in a random spot
  in contrast to ‘Buy’ this will allow for the same type of item to be bought
  It is not completely random, in the sense that it will never be an item you dislike

- **Move item**
  Use this to get colors in the positions that match the ideal room

- **Inspect item**
  get detailed info and statistics of an item in the room

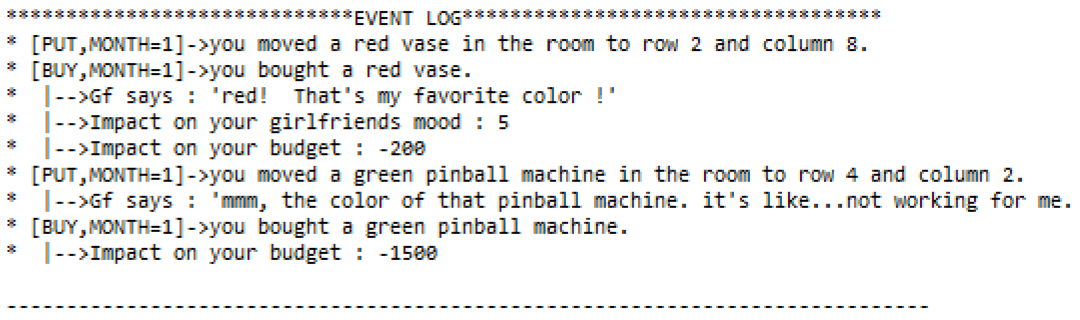
- **Score History**
  a logging of your score (and that of your girlfriend)



```
******************************SCORE HISTORY*************************************
*                 |LikePts   |EventPts  |MatchPts  |Budget    ||Total Points/Mood*
* MONTH: 2   |YOU>|35        |2         |   ---    |2214      ||37              *
*            |GF> |-25       |-4        |-20       |500       ||51              *
 ------------------------------------------------------------------------------
* MONTH: 1   |YOU>|35        |0         |   ---    |1967      ||35              *
*            |GF> |-25       |0         |-20       |250       ||55              *
 ------------------------------------------------------------------------------
* START GAME |YOU>|0         |0         |   ---    |2000      ||0               *
*            |GF> |0         |0         |0         |0         ||100             *
********************************************************************************
```

-> **likepts** : this is the sum of the likepoints of all the objects in the room (for a guy or girlfriend)
-> **eventpt**s : this is the sum of the impact of the events (see also Log)
->**matchpts**: these are the points given for the match between the room and the ideal room (so only applies to the girlfriend)
->**budget** : money you, or your girlfriend can spent
-> **total points/mood** : your total score, and the mood of your girlfriend. The mood can never be greater than 100. It can fall below zero. If so, there is a chance she will leave you.

- **Log**
  a reverse chronological list of all the events



```
******************************EVENT LOG************************************
* [PUT,MONTH=1]->you moved a red vase in the room to row 2 and column 8.
* [BUY,MONTH=1]->you bought a red vase.
*  |-->Gf says : 'red!  That's my favorite color !'
*  |-->Impact on your girlfriends mood : 5
*  |-->Impact on your budget : -200
* [PUT,MONTH=1]->you moved a green pinball machine in the room to row 4 and column 2.
*  |-->Gf says : 'mmm, the color of that pinball machine. it's like...not working for me.
* [BUY,MONTH=1]->you bought a green pinball machine.
*  |-->Impact on your budget : -1500

-----------------------------------------------------------------------------
```

-> use the comments of the girlfriend to learn about the colour preferences of your girlfriend

- **NEXT MONTH**
  proceed to the next month.
  - All scores will be recalculated,
  - if you purchased something your account will be charged,
  - a part of your salary will be added to your budget,
  - new events will generate their impact,
  - your girlfriend will comment on some items
  - …

- **End game**
  your end score will be fixed.   A new game will start

- **Quit**
  exit the game

## The classes

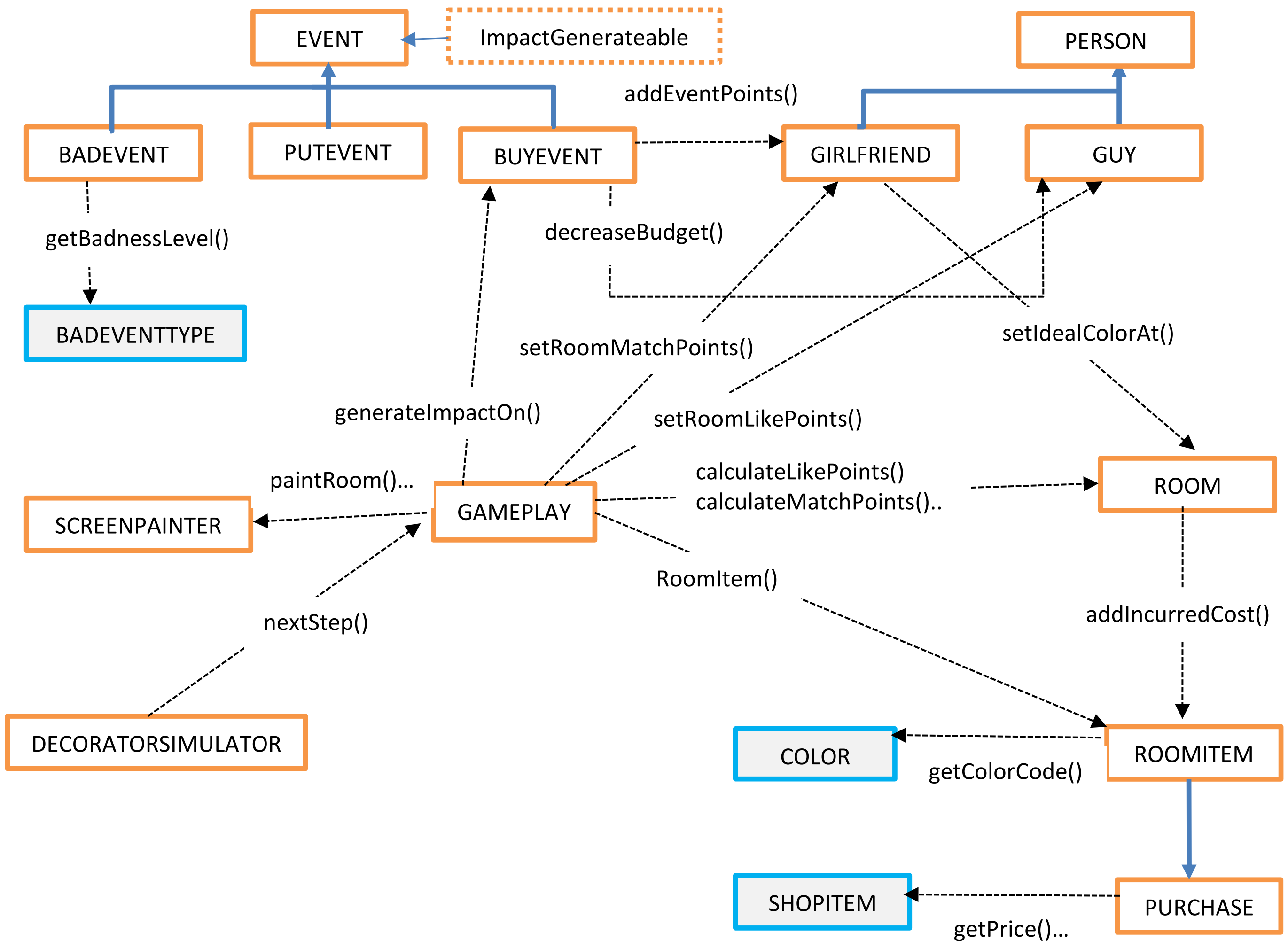
| | |
|---|---|
| DecoratorSimulator | The entry point of execution.  Not much more,.  it creates a new gameplay object and it provides a loop around the nextStep() method of the gameplay object |
| GamePlay | The most important class of the package.  It directs the flow,  keeps track of all data related to current game,  generates events, instantiates all other classes...  It also takes care off all user-interaction.<br>Each time the user enters a new game, a new gameplay object will be created.<br>The key method of the class is nextStep(), which forces the gameplay to proceed. |
| ScreenPainter | A helper class of GamePlay that builds the (non-trivial) console-output.  E.g. paint the options menu |
| | |
| Person <abstract> | A person can participate in the game.  A person has a budget to buy things and can score points. (roomLikePoints, eventPoints).   These points are also recorded in a log (histScore) |
| Guy | The player of this game takes on the role of a 'guy' .   The only added functionality in this class is the method to calculate the total points (which is simply the sum of roomLikePoints and EventPoints) |
| Girlfriend | The second protagonist in the game.  A girlfriend has a certain colour preference and preference for empty space (prefSpaciness).   Based on this, an 'ideal room' will be created in the beginning of the game (colorIdealGrid). This represents the ideal room in the mind of the girlfriend.   The more the actual room matches the ideal room, the more matchpoints the girlfriend will earn. (uplifting her mood). |

| | |
|---|---|
| | Like the guy, the girlfriend also has a method to calculate the total points. The total points of the girlfriend are referred to as 'mood'. |
| | |
| Purchase<abstract> | A purchase is something that has been bought by a person. In this version of the game these are always objects, room items that are placed in the room. But one could also include planned activities for example, like 'going to the movies'. Every purchase is linked to a ShopItem (=enum class) and the type and characteristics of the purchase are to be found within this ShopItem enumclass.<br>A purchase is linked to a buyer, and a month of purchase.<br>In order to support the 'inspect' option, extra info is kept :<br>• the log of events involving this purchase,<br>• points gained for guy<br>• points gained for girlfriend<br>• incurred cost (EUR) |
| RoomItem | A purchase that has a physical presence.<br>Every room item has a certain colour, and all colours are allowed for all room items. |
| ShopItem<enum> | All the different items that can be bought. All the characteristics of these items are taken up as the values of this enum |
| Color<enum> | All the different colors for the room items. (black, red, green) |
| | |
| Room | A room is a grid of certain dimensions (to be picked by the user).<br>At every position a room item can be placed (itemGrid), and most methods involve the manipulation of these room items in the room (retrieving, placing at a position, moving, removing, checks..)<br>A room also has a colorIdealGrid. This represents the ideal room in the head of the girlfriend (so every position can have a certain Color)<br><br>A room is able to calculate the roomLikePoints (based on the likePoints of the room items) and also the roomMatchPoints (based on comparing the colors with the ideal room)<br><br>A room can calculate the total maintenance cost for all room items(and update the incurred cost for every room item seperately) |
| | |
| Event <abstract> | An event is triggered by the gameplay. This can be a user action (buying something) or be due to some other circumstance (e.g. going in debt). It could also just be created by random chance. An event implements ImpactGenerateable, which means it generates an 1-time impact on the guy or girlfriend (when proceeding to the next month). These impacts give rise to the event points of a Person.<br>Each event has a comment field. This comment field can be used by the player to figure out the ideal room configuration<br>Each event keeps track of all the impact it has created (on points or budget) |
| BuyEvent | Whenever something is bought a buyevent is created. It is linked to a purchase and a person (the buyer)<br>1 impact will always be the impact on the budget |
| PutEvent | Whenever an item is put inside the room in a new location a put event is created. This is the case after buying the item, or after moving it.<br>It is linked to a person(the mover), a roomitem, and a location.<br>The comment on this event will give feedback to player about the positioning of the item |

| | |
|---|---|
| BadEvent | Certain ‘bad’ things like going into debt give rise to a bad event. There is a limited set of bad events described in the BadEventType enum class. |
| BadEventType<Enum) | The types of bad event that can occur. Each type has a certain badness level, which will influence the impact is it will have on the Person |
| ImpactGenerateable <Interface> | The interface implemented by Event. It has 2 methods<br>• generateImpactOn(Guy guy)<br>• generateImpactOn(Girlfriend gf) |
| | |

## Relationships between classes

In the diagram below I list all the classes and their inheritance relationships.
I also show some method calls for ilustration (so not complete)
The squares with the blue outline are enum classes, the dashed box is in an interface



## Strengths and weaknesses of the project

Strengths

- lots of possibilities to expand the game
  - the events can be used in many ways. They give you a way to generate an impact with a delay. You could even let events trigger other events etc.
  - the girlfriend can be more active in the game. Or other protagonists could enter the game. (the stepmother ?).
  - the Purchase class could be expanded with Activities as a childclass (then you could buy a romantic dinner for example).
  - …
- a lot of concepts from the lectures are used.
  - overloading (e.g. generateImpactOn())
  - overriding (e.g. toString())
  - polymorphism
  - encapsulation
  - inheritance
  - abstract classes
  - interface (ImpactGenerateable)
  - enum (e.g. ShopItem)
  - annotations and documentation
  - IO (userinput via scanner)
  - …
- I tried to avoid redundancy and give the organisation of the code enough thought. So I feel that this code is fairly easy to maintain

weaknesses

- the gameplay should be finetuned/expanded
  - Better feedback to the player to guide the game
  - All the scores and points should be scaled better
  - Probably one can find loopholes in the gameplay to maximize the score
  - Some features are not developed, like room items that span more than 1 position in the room
- Even though I did quite a lot of end-to-end tests (by playing the game), I did not have not enough time to do separate unit tests.
- The error handling should be further developed
- Some design choices don't feel quite right. I mainly think here of the hardcoded separation between guy and girlfriend (e.g. PointsGainedForGuy vs PointsGainedForGf). I did this for pragmatic reasons, but a more generic approach would have been better I think.

## Difficulties faced

Some difficulties I faced while developing this game.

- The identification of classes
  - Should something like gameplay be a class ? And a helper class like ScreenPainter ?
  - Is a room a class ? there is only 1 room in the game, so I could have incorporated the room in the GamePlay object.

- Can I put Guy and Girlfriend together under Person ? The guy plays a special role in the game because it is the perspective of the player, so should he get a separate 'Player' class ?
- When should something be a subclass vs when should something be an enum-value ? e.g. BadEvent is a separate subclass, but has different types listed in the BadEventType enum.

- Generic vs pragmatic
  - In my setup I always have a guy and a girlfriend. Many of the attributes are therefore separated in a guy and girlfriend component (like pointsGainedForGuy and pointsGainedForGf). This is not very generic off course. To make it generic I would replace the 2 variables with 1 Map, that links a Person to the attribute (so Map<Person,int> pointsGainedFor). This looks neater, but is a bit more unpractical, you always have to pass the Person on as key.

- Which functionality where ? some examples :
  - When an event impacts a person I have 2 options
    - person.processEvent(Event)
    - event.generateImpactOn(Person)

    so place the logic with Person, or put it with Event. I choose to put it with Event.

  - All the room items incur a monthly maintenance cost. Which class is responsible for updating the incurredCost-attribute of the room items ? I chose the Room class. That is a practical choice, my first reaction was to place it in the GamePlay class.

- Manage redundancy of data
  - If I buy an item should I copy for example the price of that item to that object, or just refer to the enum-object to retrieve the price (I choose the latter whenever possible to avoid copying too much data)

- Gameplay
  - Which scoring system to use as to get an interesting game ?
  - How should I scale all the parameters of the game to get a balanced game ?
  - How much randomness ? (randomness is fun but can be confusing as well)
  - The game should be easy to understand, and still be complex enough to be interesting
  - …

- Static class or not ?
  - E.g. ScreenPainter is a static class, Gameplay class is not. For GamePlay it makes more sense to make it non-static so you can start of every game with a new gameplay object. That way everything gets initialised neatly, and you can also refer back to previous games if you would want to.

- Some practical problems
  - Getting acceptable graphics on the console