



University of Glasgow | School of Computing Science

# **Gold Digger: a metaphor for searching behavior**

Gabriele Giordano Maria Rossi

School of Computing Science  
Sir Alwyn Williams Building  
University of Glasgow  
G12 8QQ

A dissertation presented in part fulfilment of the requirements of  
the Degree of Master of Science at The University of Glasgow

Date of submission placed here

## **Abstract**

abstract goes here

## **Education Use Consent**

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: \_\_\_\_\_ Signature: \_\_\_\_\_

## **Acknowledgements**

acknowledgements go here

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Overview . . . . .	6
1.1.1	Problem statement . . . . .	7
1.1.2	The metaphor . . . . .	7
<b>2</b>	<b>Survey</b>	<b>8</b>
2.1	Background theories . . . . .	8
2.1.1	Optimal Foraging Theory . . . . .	8
2.1.2	Information Foraging Theory . . . . .	9
2.2	Previous studies . . . . .	14
2.2.1	Searching behaviour . . . . .	14
2.2.2	Gamification . . . . .	14
<b>3</b>	<b>Design</b>	<b>15</b>
3.1	Site Map . . . . .	15
3.1.1	Home, Registration and Login . . . . .	16
3.1.2	World Map . . . . .	17
3.1.3	Game Screen . . . . .	18
3.1.4	General Store . . . . .	19
3.1.5	Leaderboards . . . . .	20
3.1.6	Tutorial . . . . .	21
3.1.7	Achievements . . . . .	21
3.1.8	About Page . . . . .	22

3.2	Walkthrough . . . . .	22
3.3	System Architechture . . . . .	25
3.4	ER Model . . . . .	26
3.5	Graphics . . . . .	27
<b>4</b>	<b>Impementation</b>	<b>28</b>
4.1	Development Methods . . . . .	28
4.2	Heuristic Evaluation . . . . .	28
4.3	Technologies . . . . .	28
4.3.1	Python . . . . .	28
4.3.2	Django . . . . .	28
4.3.3	HTML5 . . . . .	29
4.3.4	CSS3 . . . . .	29
4.3.5	Twitter Bootstrap CSS . . . . .	30
4.3.6	Javascript and JQuery . . . . .	30
4.3.7	AJAX . . . . .	31
4.4	Development challenges . . . . .	31
4.4.1	Creating mines . . . . .	32
4.4.2	Digging: sessions AJAX . . . . .	33
4.5	Testing . . . . .	36
4.5.1	Unit Testing . . . . .	36
4.5.2	Live Testing . . . . .	37
<b>5</b>	<b>Results and Data</b>	<b>39</b>
5.1	Data Logged . . . . .	39
5.2	Data Analysis . . . . .	39
5.3	Results . . . . .	39
5.4	Reflection . . . . .	39
<b>A</b>	<b>First appendix</b>	<b>40</b>
A.1	Section of first appendix . . . . .	40



# Chapter 1

## Introduction

### 1.1 Overview

Information management is becoming an essential part of our professional and personal lives. Whether it is in order to write a report, to do research, or simply to look for inspiration, we often turn to the internet, and increasingly less to paper-based media, to search for the information we need. Yet, to be able to make use of the vast resources at our disposal, it is increasingly important to be effective in finding ways to shorten the time spent searching, while maximising the amount of useful information found, or the 'rate of gain'. The aim of this project was therefore to provide insights into the strategies users apply to search for information and shed light on how they make searching more effective. There are however many different parameters that influence our rate of success when searching for information, such as our familiarity with the kind of information we are searching for, and our skill in using the searching tools themselves. The challenge and main objective of this project was to create a context in which all users can have the same starting conditions irrespective of pre-existing abilities, and subsequently analyse and investigate their searching behaviour.

To this purpose, we designed a game that would serve as environment to investigate searching behaviour while being abstracted from any potentially biasing context. This game, Gold Digger, simulates the actions and decisions users would face while searching for information, but not actually present them with an information retrieval task. This way, the game would make it possible to put users on the same starting point, since previous insight on information retrieval would not be applicable. As an experimental platform, it allows easy modification of its parameters and could therefore be used to carry out multiple experiments with different focus. In addition to this, the game also presents a familiar environment to users where they can have fun and will be motivated to use the system over a long time.

In Gold Digger, searching is represented by entering a mine. Examining one of the pieces of information we find and thereby acquiring useful information, is represented by digging and obtaining gold. The amount of gold that the player is able to extract from a given mine, is determined by their ability to decide whether the mine is still profitable to dig in or if it is better to move to a new mine instead, much in the same way we would decide to examine the list of links returned by a search engine one by one, or enter a new query instead. Because digging and moving both have a cost in time units, the player will need to find the most profitable way to act before the end of the day. Entering a mine also has a cost, so players will need to make sure to acquire enough gold before the end of one day to be able to continue mining. Players can purchase a variety of items that will allow them to gain more gold, dig or move at a lower cost or even be able to get clues on the profitability to dig in a mine. These items represent

the 'enrichment strategies' we use while looking for information, for instance having different tabs open on a browser to quickly flick through different pages. Finally, players are also able to gain special achievements for their performance and be ranked in leaderboards. Gold Digger is therefore an effective and creative representation of information retrieval that gathers invaluable information to further analyse search behaviour and can easily be adapted to evolving research aims, as well as keeping players engaged and motivated to take part. As a result, a very large amount of (completely anonymised) data on registered users' performance has been generated and is currently being processed.

Today, the ability to retrieve greater amounts of useful information in smaller amounts of time is becoming an invaluable skill to navigate both our professional and private lives. The insights Gold Digger can provide will help to improve information retrieval techniques, to optimise content design, and could make the skills needed to search for, find, manipulate, and make sense of vast amounts of information more accessible.

### **1.1.1 Problem statement**

This project proposes to analyse user's searching behaviour in a context deprived of any cues that could allow them to exploit any previous experience in the task in order to achieve an optimal information foraging behaviour. User's performance will be recorded and evaluated to see if it matches data in experiments in which this context is present. Findings could help provide insight on people's choices when faced with a task that requires the same kind of skills that are required in information foraging with none of the context that they are presented with in experiments based on the same theories. Finally this project aims at making user's experience as enjoyable as possible for two reasons. Firstly it will avoid users feeling like they are performing work, removing them further from a standard information foraging task. Secondly, in order to enhance the number of users that play the game as well as the amount of games played.

### **1.1.2 The metaphore**

# Chapter 2

## Survey

### 2.1 Background theories

#### 2.1.1 Optimal Foraging Theory

One of the most interesting studies proposed by Sineviro which can help us bring out some interesting points on searching behaviours, is the one on crows foraging on clams. In this study, Sineviro tells us about the foraging behaviour of the common crow (*Corvus caurinus*) in the intertidal. This kind of crow, has developed a technique to open clams which requires it to take a short flight and drop clams on some rocks below it to crack them open. The cost (in energy) of searching for clams, and the one of handling them is almost the same, however, the cost (in time) of performing the same activities is 4 times more expensive with regards to searching. It is then difficult to understand why crows would reject a large amounts of the clams they find, especially given the fact that searching seems to take up so much time. The reason for this behaviour is to be found in the "average net profitability of the clams as a function of sizeâ(Sineviro 2006), which can be represented by the equation:

$$\frac{\text{Energy}}{\text{Time}} = \frac{\text{Energy per clam as a function of size} - (\text{Search Cost} + \text{Handling Cost})}{(\text{Search Time} + \text{Handling Time})}$$

It is very interesting to notice that predictions on the crow's behaviour in order to maximise energy gain per unit of time according to this formula, closely match its observed behaviour.

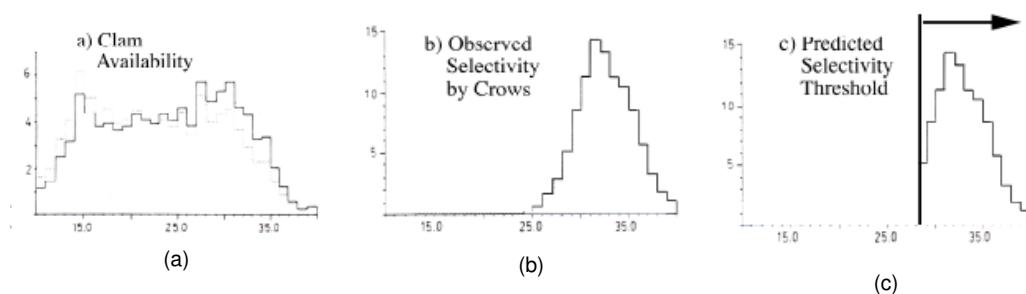


Figure 2.1: frequency of clams in class size (% on y axis) by clam length (x axis)

As we can see from (b) and (c) in fig 2.1, predicted behaviour seems to closely match observed

behaviour. This kind of model, however, considers the disposition of prey in the environment to be random but, sometimes, prey items seem to have a patchy distribution (Sineviro 2006). In this case, an animal will have to travel between patches before it can start exploiting a new one. This creates two variables according to which an animal has to "make its decision" in order to maximise its rate of gain. The first one is the time spent within a patch to feed, and the second one is the time spent travelling between patches in which, crucially, the animal doesn't gain any energy and instead consumes it. A formula that effectively models this choice is:

$$\text{Rate of energy gain} = \frac{\text{Energy}}{\text{Time}} = \frac{\text{Energy or Load Size}}{(\text{Travel time to patch} + \text{Foraging time to patch})}$$

Also known as "**Charnov's Marginal Value Theorem**" which generates an energy gain function like the one on the left. Here, we can see how there are two main areas delimiting the Cartesian space, the first one (on the left side of the curve) is the potential time the animal has to spend in between patches, whereas the second one is the potential time spent feeding in a single patch. An optimal allocation of time in foraging for this rate of gain will be represented by the tangent to the curve as shown in the graphs on the right. The red line is the tangent and the point of intersection with the curve determines the in-patch time that would be optimal for an animal to spend feeding.

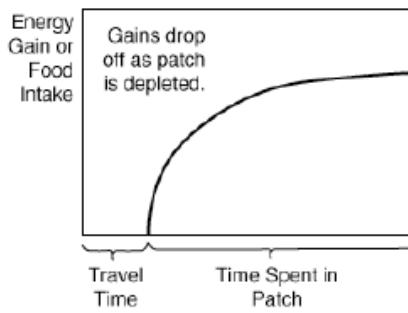


Figure 2.3: An energy gain function

The aim of this proposal is evaluate subjects' innate ability to reproduce similar behaviours while foraging for information rather than for food. To do this, the context of a gold digging strategy game has been chosen, in order to remove all proximal cues that might depend on the user's previous experience. In this environment, the user will have to perform the same kind of cost vs. benefit choices, however, he would not be able to adopt the strategies he consciously would. Following I discuss a paper by Pirolli & Card on Information Foraging which seems to support this intuition proposing experimental evidence and mathematical models. "

### 2.1.2 Information Foraging Theory

Pirolli & Card seem to start from the assumption that there are many similarities between the techniques adopted by animals foraging in the wild and the ones adopted by people in "Information Foraging". Because of the structure of today's society, in fact, it is not for food that we forage for. Food is usually readily available almost anywhere human settlements can be found, however, to purchase it, we need to engage in a "complex tributary of cultural tasks that engage our physical and social environments" (Pirolli & Card 1995) which demand us to develop numerous "information-based" strategies, in order to earn a living. Pirolli & Card begin by explaining that our adaptive success seems to be increasingly dependent on our mastery of techniques of information-gathering, sense-making, decision-making and problem-solving strategies. In turn, they claim, this leads us to implement strategies which are similar to the ones seen in foraging

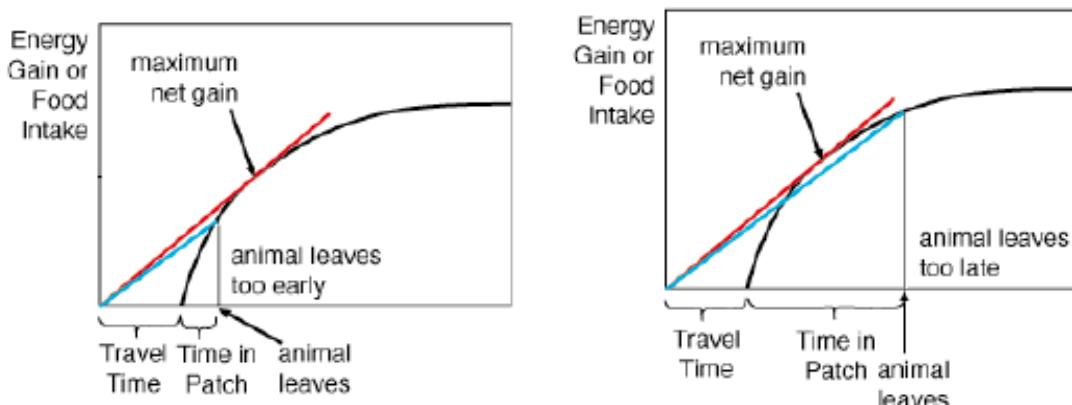


Figure 2.4: The optimal rate of gain

behaviours of animals in the wild. Here, we can observe patterns in food foraging behaviours which aim at maximising the energy intake while minimising the amount of energy spent while foraging.

In other words, our lives are increasingly dependent on the ways in which we organise and retrieve information in order to use it effectively to navigate and survive today's social (and sometimes physical) landscape. Pirolli & Card go on to claim that, the Information Foraging theory's main tenant is that: "when feasible, natural information systems evolve towards stable states that maximise gains of valuable information per unit cost. Cognitive systems engaged in information foraging will exhibit such adaptive tendencies" (Pirolli & Card 1995).

As a consequence, it is to be expected that people will modify their information foraging strategies as well as the structure of the environment itself, whenever possible, to "maximise the rate of gaining valuable information" (Pirolli & Card 1995). Better strategies will be the ones that will allow someone who adopts them to yield more information per unit cost. Furthermore, it is also expected that these strategies will evolve through time, in order to reach a seemingly stable state that maximises the potential gains.

The process of analysing the development of this kind of behaviour, is called "adaptation analysis". This kind of analysis is conducted here, as well as in biology, through the use of "optimisation models" to study the design features of organisms and artefacts. Optimisation models include three major components:

- *Decision assumption*, determine how much time is to be spend analysing a certain collection of information as well as which kind of content is worthwhile pursuing.
- *Currency assumptions*, determine how the choices made through decision assumptions are to be evaluated. In the context of Information Foraging, the relevant currency will be amount of relevant documents found, as opposed to the amount of energy gained in food foraging strategies.
- *Constraint assumptions*, determine what kind of limits will apply to the relationship between decision and currency assumptions. In Information Foraging, these include (but are not limited to) previous knowledge, available technology and task structure.

These models allow us to construct a framework for the evaluation in Information Foraging, however, it is not to be expected that any single individual will fully be conscious and even evolve towards, an optimal awareness and implementation of these models. These models simply outline the possibility of "an advantageous adaptation if not blocked by other forces".

The main decision an organism has to make in its foraging endeavours (being it for food or for information) is determined by a problem of "Enrichment vs. Exploitation" of a certain "patch of relevant documents (or food). The careful weighing of one against the other, will, in turn, determine its searching behaviour, comprehensive of "in-patch" and "between-patch behaviours. When we decide to "enrich" a certain patch of information, we engage in certain "enrichment strategies" which are meant to maximise the amount of relevant information per unit of cost that we are able to get from a patch. However, the adoption of these strategies themselves has a cost which should be considered when making the decision to move to a different patch and select a new set of documents. There are two main "in-patch" enrichment strategies.

The first one is based on producing information packages that yield better results. This involves strategies like developing or acquiring better search tools, as well as spending time mastering them. The second one is based on filtering the incoming information into relevant topics or according to other decisional criteria devised by the foragers and that they came to realise as useful to their foraging needs.

Another way foragers can increase their gain in yielded by their foraging behaviours is by using "between-patch" enrichment strategies. These are also of two kinds. The first one is constituted by what Pirolli & Card call "scent-detection strategies". These strategies are based on the identification of proximal cues in the environment in order to make a choice on whether it would be fruitful to explore a certain "patch" or move to another one based on detection of proximal cues. In the context studied by Pirolli & Card, this translates into identifying the relevance of a certain document, or group of documents by, for instance, its title and payoff, perceived clarity of writing or length.

A second kind of in-patch enrichment strategy is based on reducing the cost of getting from one information patch to another. While this is usually impossible for animals in the wild, because it involves modifying the environment, it is instead a very efficient way for information foragers to improve the rate of gain per unit cost. In the example proposed by Pirolli & Card this is shown in the re-organisation of the workspace of an employee whose job is to write a Business Intelligence Newsletter. The subject of their experiment, in fact, organised the different areas of his office, in order to minimise the time spent searching for relevant document by disposing the material he knew he would need more frequently, the nearest to his working station.

For the purpose of this proposal it is also important to consider the conventional models of foraging on which Information Foraging Theory is based. As it can be expected, to provide an efficient model of foraging, we will have to take into account equations which model both in-patch and within-patch behaviours. Given what previously stated, Pirolli & Card start by characterising the rate of gain of valuable information per unit cost  $R$  as the ratio of the total net amount of valuable information gained  $G$  divided by the total amount of time spent between patches  $T_B$  and exploiting within patches  $T_W$  a

$$R = \frac{G}{T_B + T_W} \quad (2.1)$$

Notably, this equation assumes that (1) "the total amount of information gained can be represented as a linear function of between-patch foraging time:

$$G = \lambda T_B g \quad (2.2)$$

And that (2) the total amount of within-patch time can be represented as:

$$T_W = \lambda T_B t_W \quad (2.3)$$

This, in turn, gives Holling's Disc Equation:

$$\begin{aligned} R &= \frac{\lambda T_B g}{T_B + \lambda T_B t_W} \\ &= \frac{\lambda g}{1 + \lambda t_w} \end{aligned} \quad (2.4)$$

This formulation, however, "addresses the problem of allocation of time between in-patch vs. between patch under certain strong assumptions". Because of this, Pirolli & Card have to devise their own interpretation which takes in to account that "(a) there might be different kinds of patches and (b) the total gains from a patch depend on the within-patch foraging time which is under the control of the forager. For this reason they have to include a value  $i$  representing the type of patches encountered at a rate of  $\lambda$  i and a value  $t_{wi}$  representing the *policy* adopted by a forager on how much time to spend within each patch. The total gain can then be represented as the sum of all the values of  $i$  between 1 and P.

$$\begin{aligned} G &= \sum_{i=1}^P \lambda_i T_B g_i(t_{wi}) \\ &= T_B \sum_{i=1}^P \lambda_i g_i(t_{wi}) \end{aligned} \quad (2.5)$$

In the same way, the total amount of time spent within patches could be represented as:

$$\begin{aligned} T_W &= \sum_{i=1}^P \lambda_i T_B t_{wi} \\ &= T_B \sum_{i=1}^P \lambda_i (t_{wi}) \end{aligned} \quad (2.6)$$

Combining these two equations according to (eq. 2.1) gives us:

$$\begin{aligned} R &= \frac{T_B \sum_{i=1}^P \lambda_i g_i(t_{wi})}{T_B + T_B \sum_{i=1}^P \lambda_i t_{wi}} \\ &= \frac{\sum_{i=1}^P \lambda_i g_i(t_{wi})}{1 + \sum_{i=1}^P \lambda_i t_{wi}} \end{aligned} \quad (2.7)$$

This last equation is what Pirolli & Card call the "patch model for information foraging" which takes into account patches that yield different kinds of gain functions as well as different strategies to exploit them.

Figures (a), (b) and (c) are the graphical representation of the functions presented in the previous section according to Charnov's Marginal Value Theorem, they model the problem of allocation of time between in-patch vs. between-patch strategies. Here, between-patch search times  $t_B$ ,  $t_{B1}$  and  $t_{B2}$  generate different tangents to the  $g(t_w)$  gain function which, in turn, determines the optimal rate of gain  $R$ . The outside of the curve on the x axis represents between-patch time while, the inside of it, represent within-patch time. To determine the optimal rate of gain  $R_*$  one draws the tangent line to the gain function  $g(t_w)$  and passing through  $t_B$ . The point of tangency will determine the "optimal allocation of within-patch foraging time  $t_*$ ".

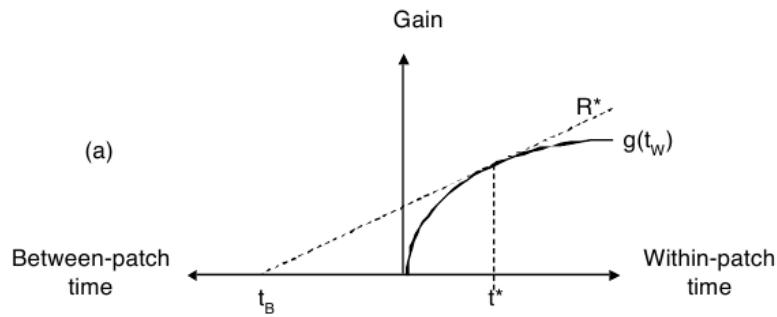


Figure 2.6: (a) - The Graph illustrates Charnov's Marginal Value Theorem where  $t^*$  denotes the optimal amount of time spent within-patch given the intersection between the gain function  $g(t_w)$  and the tangent passing from  $t_B$  (the time spent between patches)

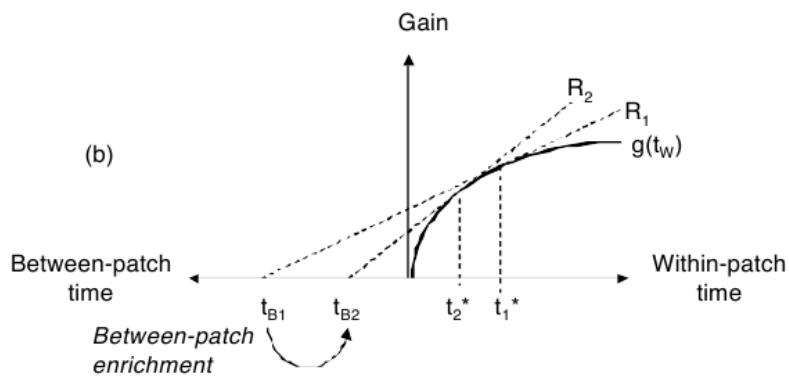


Figure 2.7: (b) - Shows the changes generated by the adoption of between-patch enrichment strategies. The tangent to the gain function  $g(t_w)$  passing from  $t_{B2}$  determines that the optimal amount of time to spend in a given patch is now  $t_{2*}$ . The forager is now able to afford to spend less time in each patch.

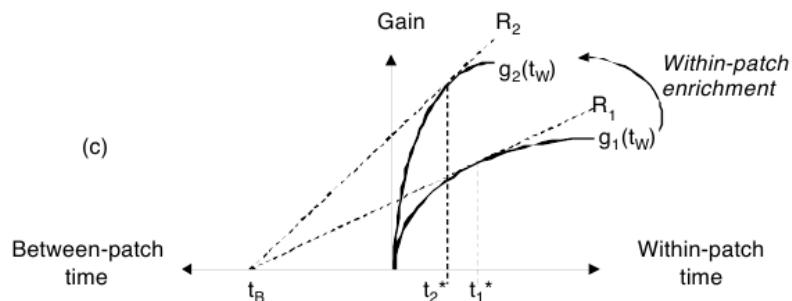


Figure 2.8: (c) - Shows the changes generated by the adoption of within-patch enrichment strategies. The new gain function  $g_2(t_w)$  has a higher rate gain which allows foragers that spend the same amount of time between patches  $t_B$  will be able to reap more results in a shorter time.

This means that, the more time is spent between-patch, the less a forager will find it beneficial to spend time exploiting a given patch, always depending on the gain function  $g(t_w)$ . In Figure 2, in fact, we see the effects of a between-patch enrichment like the one described previously (office re-disposition) which sets the optimal rate of gain in such a way that it will be more beneficial for the forager to spend more time within-patch to exploit it. Finally, in Figure 3 we can see the effect of within-patch enrichment (for instance, making information packages that yield better results). As we can see, within-patch enrichment, changes the gain function  $g(t_w)$  determining higher gains per unit of time spent within-patch.

The objective of the experiment outlined in this proposal, is to be able to evaluate the strategies

adopted by subjects in order to reach an optimal rate of gain and to analyse how they are able to choose between within/between-patch enrichment strategies to better their rate of gain per unit cost.

## **2.2 Previous studies**

### **2.2.1 Searching behaviour**

### **2.2.2 Gamification**

# Chapter 3

## Design

The design of Gold Digger went through different iterations through the course of its development (see 4.2 Heuristic Evaluation), however some design choices were made in order to keep the user experience consistent throughout the website and enhance clarity and ease of use.

### Separation between "game pages" and "site pages"

Because Gold Digger is both a website and a game, it seemed appropriate to somehow separate the "game environment" from the more "site-like" features and pages, while still maintaining a certain degree of coherence between them. "Game pages" are the ones immediately relevant to the game in itself, the ones that the user will most likely enter while playing Gold Digger. These are: **the Game/Mine page, the General Shop page, the Home Page and the World Map Page**. The other pages are said to be "site pages" and they include pages like the About page and the Leaderboards, which a user is not very likely to access while in the middle of a play through. Furthermore "game pages" all require the user to be logged in to be accessed while the "site pages" do not.

Notwithstanding this distinction, the user experience is not disjointed thanks to prominent recurring elements such as the nav bar on the top of the page and the landscape headline showing the name of each page.

### 3.1 Site Map

Navigation through the website is aided by a navbar located at the top of every page (with the exception of the 'game over' and 'end of the day' pages).

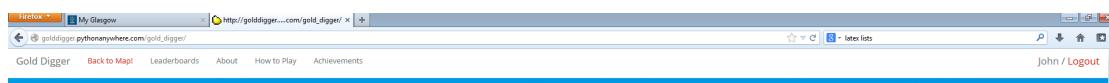


Figure 3.1: Gold Digger nav bar

From the navbar the user will be able to reach the following pages:

- The main page (by clicking on 'Gold Digger')
- The World Map page (if signed in)

- The Leadeboards
- The About page
- The 'How to Play' / Tutorial page
- The 'Achievements' page
- The User Profile page (if signed in)

finally, the user will also be able to logout at any moment by using the 'Logout' button on the top right. However, if the user is currently in one of the mines, clicking on one of these links will result on a warning message being displayed that will alert users that the gold gathered during the day will be lost if they leave the mine before exhausting the time at their disposal and reaching the end of the day.

### 3.1.1 Home, Registration and Login



Figure 3.2: Home Page

On the landing page (homepage) users will be presented with a short **five-point explanation** of the game to quickly explain the mechanics of the game so that users could start playing as soon as possible. To do this, they will have to login through the form on the left or register by clicking on the 'Register' button.

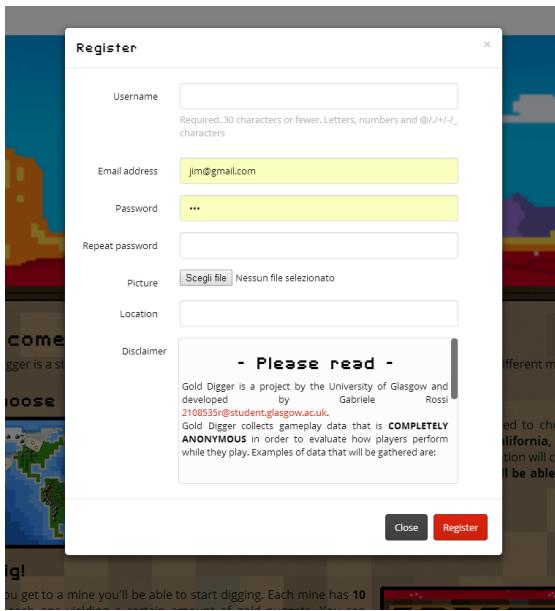


Figure 3.3: Registration modal

Clicking on the '**Register**' button will trigger a modal asking users to create a new username and password, as well as optionally entering their location and user picture. Finally, the modal displays a disclaimer in order to both make sure that the users are aware of the limitations of accessing the website and its purpose. Users who wish to have more information about Gold Digger are redirected to the 'About' page or offered a link to directly write an email to the developer.

Finally if a user enters wrong details or forgets to fill in a required field, (both for registration and login) an appropriate error message is clearly displayed for the user to see and try again. At present there is no limit to the number of attempts a user can make at logging in or registering.

### 3.1.2 World Map

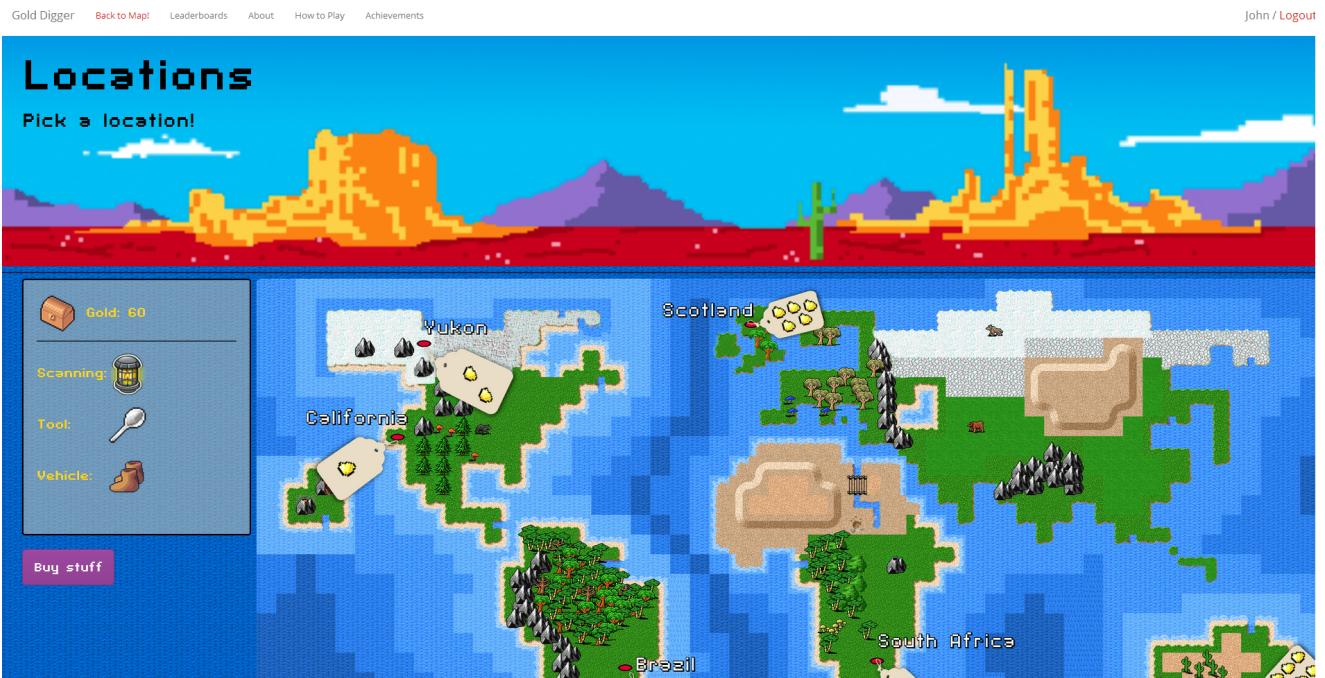
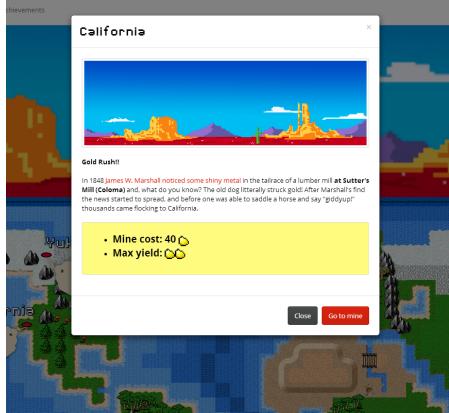


Figure 3.4: World Map

Once users have registered or have been logged in, they are automatically redirected to the world map so that they can start playing immediately. On the left of the page, users can see the equipment and the amount of gold they have at the moment. The side item panel appears on the left column of the page, once the user has logged in. The panel shows the items that users have equipped, together with the total amount of gold in their possession. By hovering over each one of the items a tooltip will appear, showing the essential stats of each item. The

presence of the side item panel has the function of both confirming the users that they are logged in and reminding them that their game session has started contributing to the uniformity of the user experience.



On the right hand side of the page users can access any of six locations: **California, Yukon, Brazil, South Africa, Scotland** and **Victoria**. Clicking on any of the locations will trigger a modal that displaying the scenery of the particular mine, some information about the gold digging history in that region (with links to Wikipedia articles), the cost of the mine and the amount of gold the user can expect to find. Starting from California and ending with Victoria, each mine is more expensive than the previous one but it also have a potentially higher gold yield. It is up to the player to enter the right mine at the right time.

Figure 3.5: California modal

### 3.1.3 Game Screen

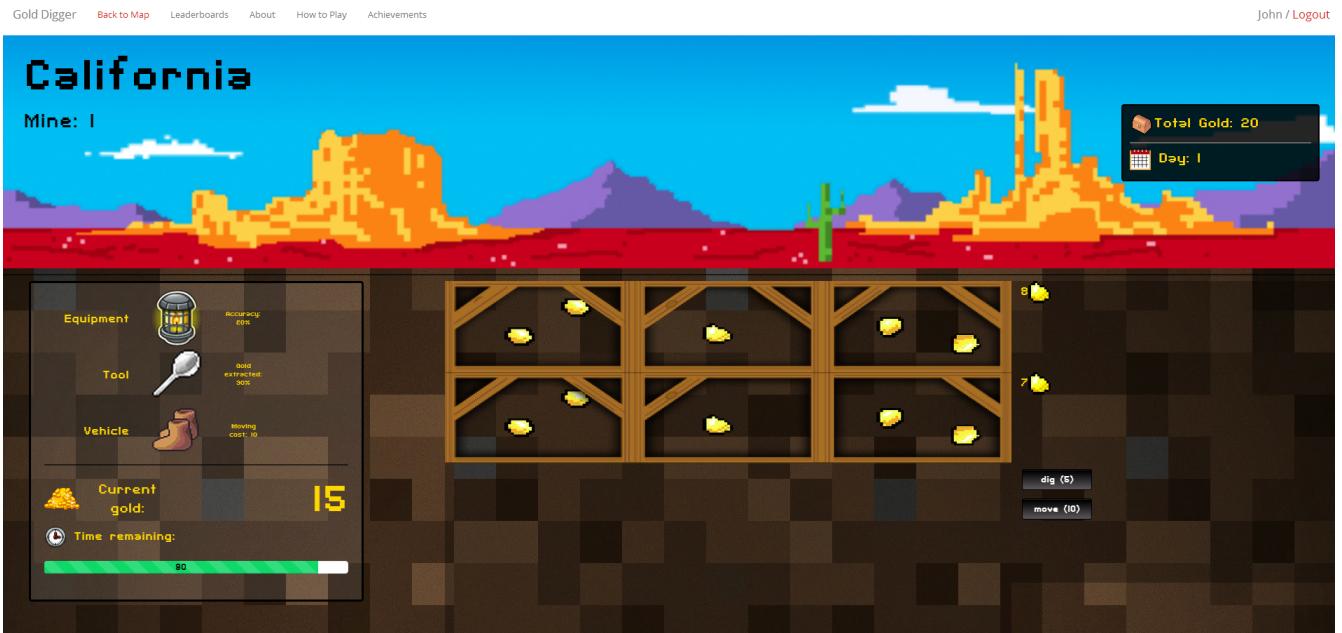


Figure 3.6: Mine

Once the users enter a mine, the appropriate amount of gold is removed from the total and they can start digging. Each mine presents a similar structure, except for the landscape and the amount of gold that can be found in each layer. On the top of the page we can see the landscape picture and the name of the location, as well as the number of mines that the user has dug into during the present day. As previously noted, on the left, we can find the equipment panel, in the middle we have the mine shaft and on the right have the '**Dig**' and '**Move**' as well as the yield of each layer that has been already dug. The left side panel contains here some more information than it does in the other pages where it appears:

- **Current gold:** the amount of gold gathered during the present day.

- **Time remaining:** the amount of time (in units of time) remaining before the end of the day.

Furthermore, because the left side panel's position is fixed, it will follow users as they get deeper and deeper in the mine, allowing them to always keep an eye on the game state without having to go back to the top of the page to check how many units of time they have left before the end of the day. Because the 'Current gold' is summed to the 'Total Gold' only at the end of the day, the total amount of gold, together with the number of days users have been digging without encountering a game over, is displayed in a small box on the top right of the page. The options available to the user and an example of gameplay are detailed in section 3.2.

### 3.1.4 General Store

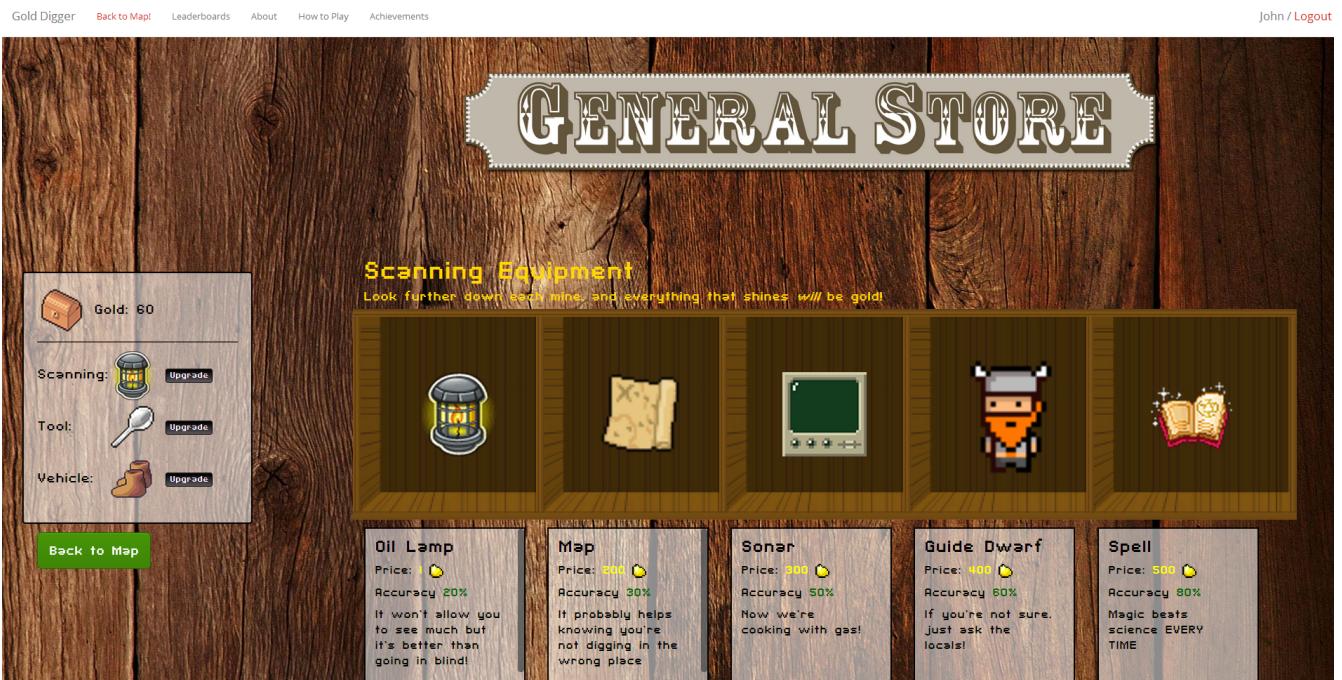


Figure 3.7: General Store

At the end of each day (by clicking on 'shop'), or from the world map screen (by clicking on 'Buy stuff'), users can access the **General Store** (see fig. 3.7). From here they are able to browse and purchase new items to help them in their digging. There are three groups of objects that the user can choose from **Scanning Equipment**, **Digging Equipment** and **Vehicles**. Each of these groups contains five different items with different cost and stats. Users are able to see the image associated with each item as well as its cost, stats and a short description in the small tex box underneath. If users decide to upgrade one of their item groups (Scanning, Digging or Vehicle) they can do so by clicking on the '**Upgrade**' button. At this point they will be presented with a small modal that reviews the item's stats, together with its picture, asking the user to confirm the purchase. If the users don't have enough gold to make the purchase, or if the purchase would cause them to not have enough money to enter the cheapest mine, an appropriate alert message is displayed, explaining why the purchase is not possible.

The choice to allow users to *upgrade* rather than *buy* items has been made because each of the items is better than the previous one in every respect and thus there is no reason why a user would chose to equip an item that they previously purchased, since this would not bring

any advantage at all. For instance, with digging tools, their cost, dig cost (the amount the user has to pay in time units to dig once) and extraction power (the percentage of gold that the user will be able to extract given a certain yield), are all increased from the least, to the most expensive. However, it would be easy to add items that have different combinations of these parameters and add, for example, a digging tool that, at a higher cost per dig also returns a higher percentage of gold. Finally, upon purchase, the new item is immediately added to the left side panel and the appropriate amount of money removed from the user's total through an AJAX call.

### 3.1.5 Leaderboards

Average	Max Gold	Days Worked	All Time Gold	Achievements	Average Gold Per Mine	All Time Gold Dug	Max Gold Reached	Days Dug
Rank	Thumbnail	Username	Equipment					
1		Garedyen			272.0	83602	56060	60
2		koocharrs			271.0	97392	71113	77
3		Tina			197.0	32187	9964	58
4		s.andi			163.0	16135	3622	32
5		luke			159.0	14532	3030	28

Figure 3.8: Leaderboards

On the **Leaderboards** (see fig. 3.8) page, users are able to check their performance and compare it to the other players'. There are four parameters by which users can be ranked, plus an achievements board that has no specific ranking criteria:

- **Average:** the total amount of gold ever dug by the player divided by the total amount of mines that were dug into.
- **Max Gold:** the maximum amount of Gold ever dug (the maximum amount of gold ever reached)
- **Days Worked:** the total amount of days of digging completed by the player (not zeroed on game over)
- **All Time Gold:** the cumulative total amount of gold dug by the player (not zeroed on game over)
- **Achievements:** shows the achievements gained by each player in no particular order (achievements are not ranked)

Each row of the the tables (excluding the 'Achievement' table) diplays the following parameters: Rank, Thumbnail, Username, Equipment (showing the image for each of the three item groups), Average Gold per Mine, All Time Gold Dug, Max Gold Reached, Days Dug.

### 3.1.6 Tutorial

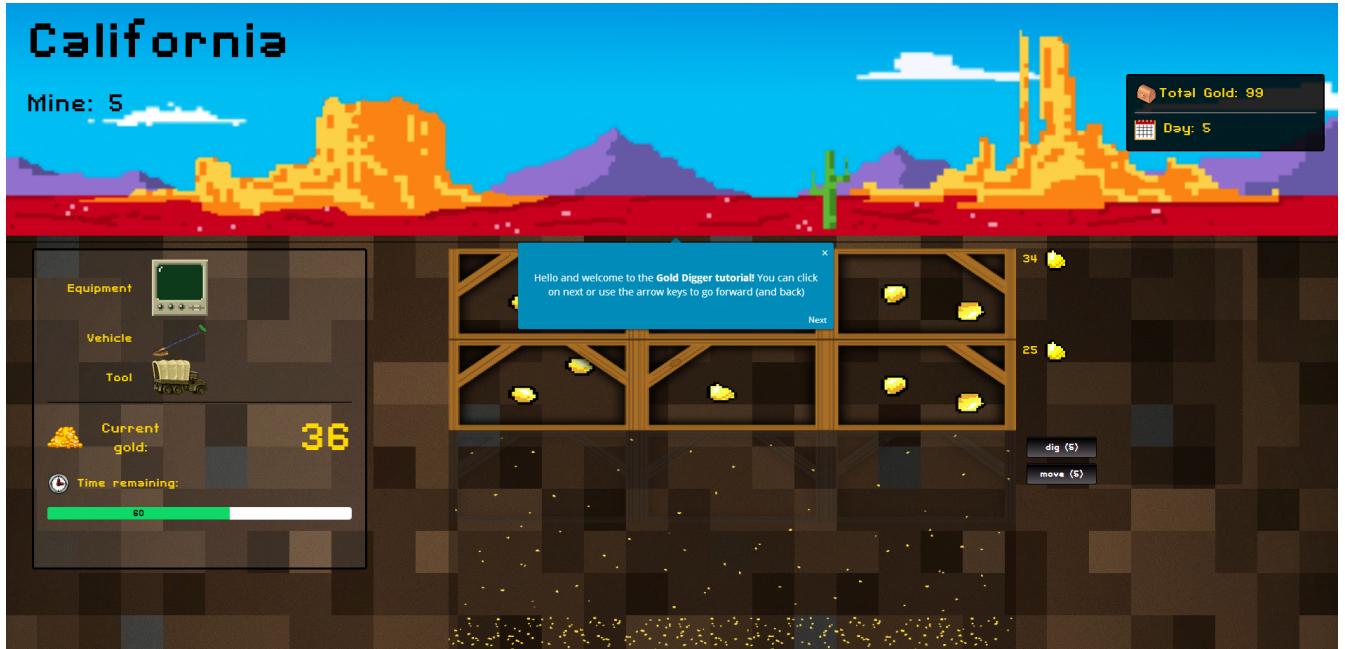


Figure 3.9: Tutorial

Because it is important for users to have a clear idea of the way the game works, all effort has been made to employ a visual approach that would be quick and easy to understand. To this end, upon entering, the '**How to Play**' page, a tour of the main game features is automatically launched. The javascript library **Trip.js** (see 4.4.6) highlights and points at the different parts of the game screen in order to get a quick visual explanation of the main game mechanics. Each of the labels has 'Next', and 'Previous' link, in order to let users go back to a previous point or skip explanations they already viewed.

### 3.1.7 Achievements

The '**Achievements**' page dispays all the achievemts badges together with their name, the condition that triggers them and their image. This page is needed in order for the users to be able to check which achievements they can aim for while playing. Two additional achievements (fig. 3.11) are not displayed here and are given only under special conditions:

- **Awesomeness**: given to the people who tested the "beta" version of the site.
- **Banana**: given to people who managed to find the game's Easter Egg

Achievements ans special achievements have been incorporated in order to both add an extra game feature to Gold Digger which wouldn't change the game mechanics too much an to give players an incentive to keep playing, especially since there is no particular "winning condition", fulfilled which a player has completed the game.

### 3.1.8 About Page

In this page, users are able to find the developer and supervisor's contact details as well as reading about the reasons behind the creation of Gold Digger and acknowledgements of the authors of some of the material used in the site.

## 3.2 Walkthrough

(1) Jill, the user, has just logged in (or registered) and has been redirected to the World Map. Her starting Equipment and Gold are as follows:

- **Gold:** 100
- **Scanning Tool:** Oil Lamp (accuracy: 20%, visibility: 2)
- **Digging Tool:** Spoon (dig cost: 5, gold extracted 30%)
- **Vehicle:** Boots (move cost: 10)

On the map, Jill clicks on California and a modal appears, from which she can see that, although the mine doesn't yield much gold, it is relatively cheap to access it (she only needs to pay 40 gold nuggets to enter it).

(2) After clicking on 'Go to Mine', 40 gold nuggets are removed from Jill's total amount of gold and she is presented with a mine where no layer has been dug. On the top left hand side Jill can see the name of the location she is in and the number of the mine (now 1) whereas on the right hand side she can check her total amount of gold (now 60) and the number of days she

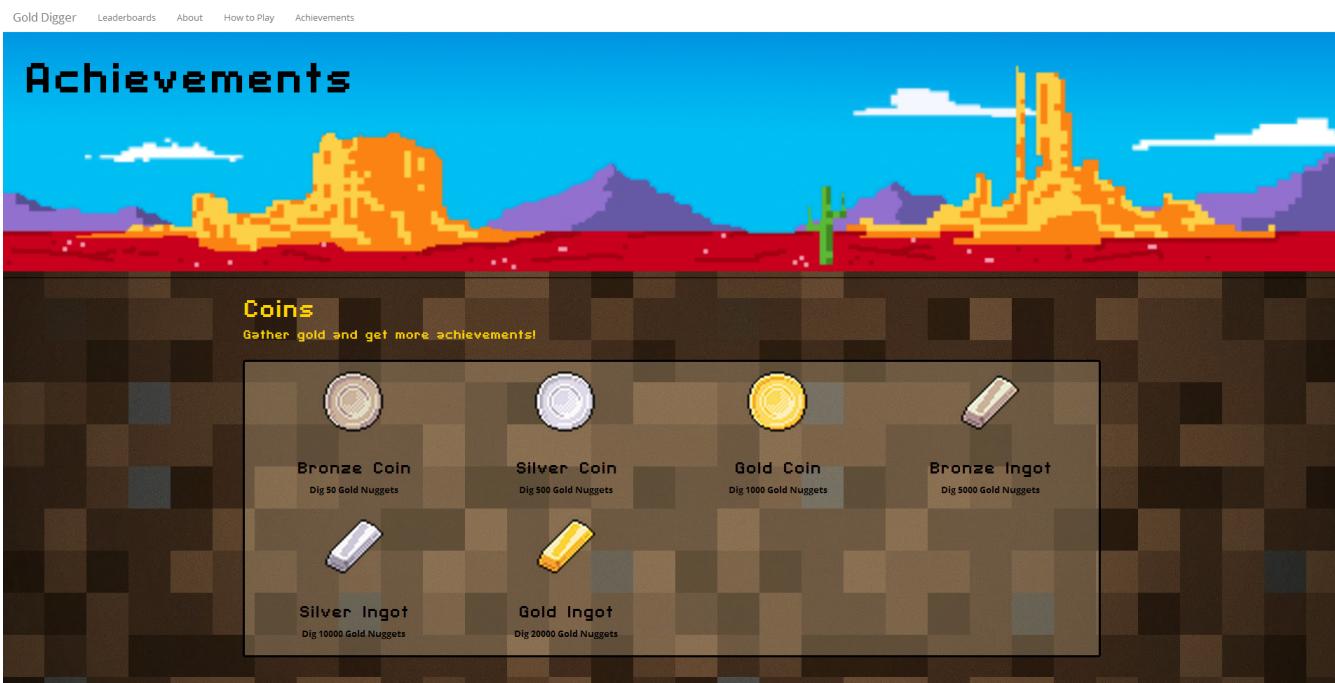
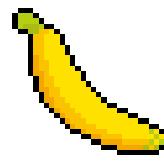


Figure 3.10: Achievements display

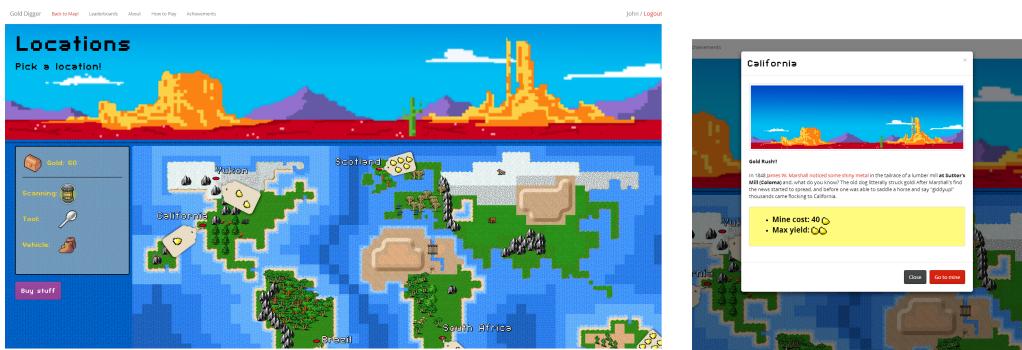


(a) You helped testing!



(b) You found the Easter Egg!

Figure 3.11: Special Achievements



has been digging. Thanks to her Oil Lamp she can see specks of gold in the first two layers but, because the lamp only has a 20% accuracy, it is possible that the amount of flecks that can be seen is deceiving.

- **Current gold:** 0
- **Time remaining!**: 100
- **Mine:** 1
- **Day:** 1

(3) Jill decides to dig through the first couple of layers layer of the mine by clicking on the 'Move' button on the right hand side of the mine shaft. She gains 17 gold nuggets and consumes 10 units of time (since each digging operation costs 5 units of time)

- **Current gold:** 17
- **Time remaining!**: 90
- **Mine:** 1
- **Day:** 1

(4) At this point Jill is not sure she might get much more gold from this mine and she doesn't want to spend precious time digging through layers of the mine that she has no information about. For this reason she clicks on 'Move' to get to a new mine (in the same location). To do this she must spend 10 units of time

- **Current gold:** 17
- **Time remaining!**: 80



- **Mine:** 2
- **Day:** 1

(5) Jill continues to dig through some more mines and after a while she runs out of units of time. At this point, she is presented with the 'End of the Day' screen that sums up her progress during the day as follows:

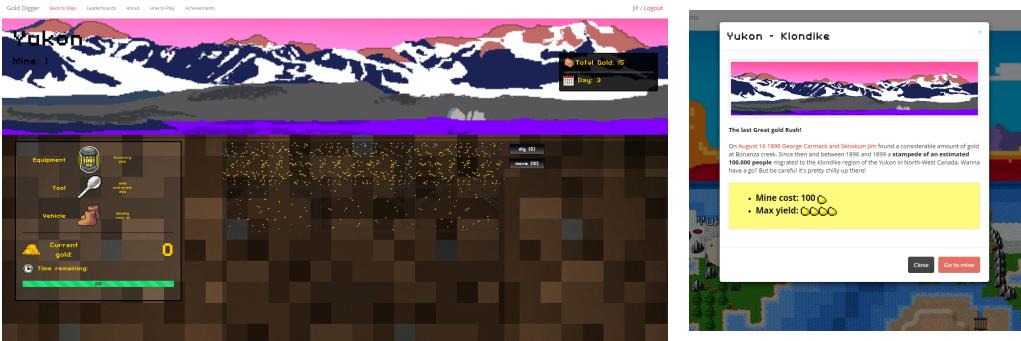
- **Today's gold:** 94
- **Total gold:** 94
- **Mines dug:** 6

Jill also receives the 'Bronze Coin' achievement for having dug more than 50 gold nuggets. From here Jill decides to go to the shop and see if she can purchase some new items



(6) Once in the shop Jill tries to purchase better scanning equipment, so she clicks on the 'Upgrade' button next to the Oil Lamp. Unfortunately she doesn't have enough gold to make this purchase yet, so a message alerts her that she would need to do some more mining if she wants to be able to purchase the item.

(7) To gather more gold, Jill goes back to the California mines and, as soon as she has 100 gold nuggets, she takes a gamble and spends them on accessing the Yukon mine. However, she doesn't perform very well and, at the end of the day she owns less than 40 gold nuggets. Because she doesn't have enough money to enter any of the mine, Jill loses her first game. However, clicking on 'Back to Map' will restore her to the initial conditions, if she bought any items they would be lost.



### 3.3 System Architecture

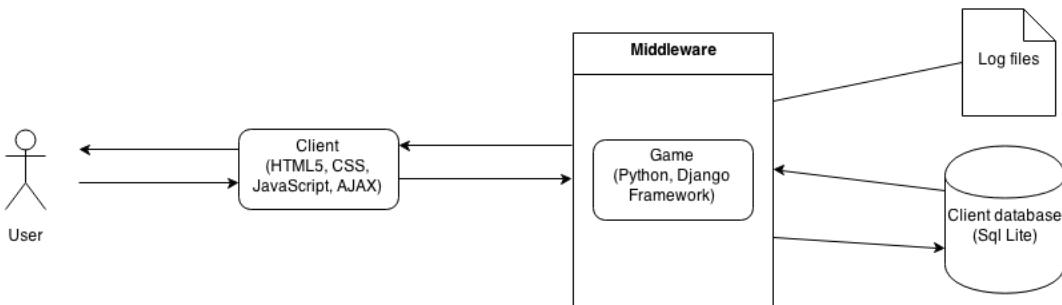


Figure 3.16: Gold Digger's System Architecture

Gold Digger is based on a 3-tier architecture. Each user is able to create a player account (client) containing all of the details entered upon registration, together with a set of game variables, set to their starting values. The client side renders HTML5 and CSS3 elements and graphics while updates to the game states are done mostly through AJAX, in order to avoid reloading the page. The game logic is coded in the middleware using the python-based Django framework (see 4.3.2). All of the user profiles as well as the other game objects and achievements are stored in the database while the corresponding graphics, as well as the rest of the game graphics are stored in the project's static and media folders. Finally, the middleware continuously updates a log file in order to monitor user behaviour and gather data to be parsed and analysed at a later stage.

## 3.4 ER Model

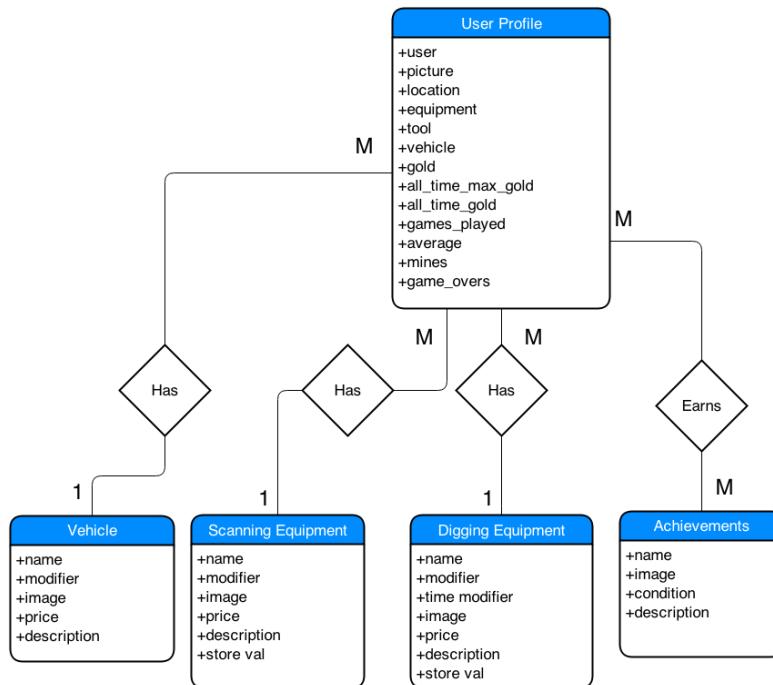


Figure 3.17: The Gold Digger ER diagram

The above diagram describes the entity-relationship model for Gold Digger. There are 5 entities in Gold Digger:

- **User Profiles:** this entity contains several of the users' game stats, like the amount of gold at their disposal and the average amount of gold dug per mine as well as a reference to the Django object 'user' containing all the information regarding the user's password, username and email. This is necessary in order to take advantage of Django's very efficient user management system and keep the website as safe as possible
- **Digging Equipment:** this entity contains each of the tools' name, image, price and description. However, the most important attributes of this entity are the *modifier* and the *time modifier*. The *modifier* attribute is used by the game mechanics to calculate how much gold a user is able to extract from a given layer of the mine. It is a *float* number between 0.2 and 0.8 that will be multiplied by each of the amounts of gold in each layer. This way if the user is using, for instance, a shovel, she will be able to extract only 30% of the gold that is actually present in that layer (see ??). The *time modifier* attribute, on the other hand stores the digging cost per digging operation of each of the tools and it is an integer that goes from 5 to 1 in decreasing order from the worst tool to the best.
- **Scanning Equipment:** this entity is quite similar to Digging Equipment but it does not include the *time modifier* attribute. The *modifier* attribute of this entity is responsible for calculating the cue array, that is in turn responsible for determining the amount of gold flecks that will be shown in each of the mine's layers (see ??). Furthermore, because this attribute holds only float values from 0.2 to 0.8, a simple multiplication by 10 is used to determine the number of layers in which those specs are actually visible .
- **Vehicles:** this entity, like Scanning Equipment, only posses the *modifier* attribute. This is an integer in range 10 to 6 and it holds the value of time units to be subtracted from the day's time, for each digging operation.

- **Achievements:** the Achievement entity simply stores the name, image, condition and description of each of the achievements.

## Relations

There are **four relations** between the above entities. Each player can only store only one relation to a particular Scanning Equipment, Digging Equipment and Vehicle object thus, these are all one-to-one relationships. However, because each user can earn many achievements, the relationship between the User Profile and the Achievement entity needs a separate database table to keep track of the user-achievement value couples. In this table, **User Achievements**, each row comprises two columns, each of which is a foreign key. The first one is the relation to a User Profile Object, and the second one is the relation to an Achievement object.

## 3.5 Graphics

One of the main elements of the design of Gold Digger was its graphic design. The general idea was to reproduce the look and feel of a retro 8-bit game as much as possible, without compromising the clarity and ease of use needed by a website. For this reason, 8-bit-style graphics were employed in most of the elements of the 'game pages' while Twitter Bootstrap was used in order to provide a simple minimalistic "framework" for the graphics to be displayed in.

### Fonts

There are two fonts that have been used throughout Gold Digger. The first one is **Open Sans** (see 4.3.5). This font is part of the Twitter Bootstrap theme that provides the graphic framework for the site. It is used in areas with lots of text and in the navbar, where clarity the most important issue. However, throughout the site, most of the headings, scores, descriptions and game messages employ the font **SF Pixelate** (by ShyFonts) which has a pixelated look reminiscent of retro 8-bit games. All of the text in the 'game pages' is typed using this font.

### Landscapes and World Map

There are six different landscapes in Gold Digger, each one associated with one of the locations that the user can travel to and dig into. Three of them (California, Scotland and Yukon) were created with a pixel-art specific (Pixelen) program using a number of pictures for inspiration, while the remaining three have been produced by applying filters, modifying and coloring similar pre-existing pictures in Photoshop CS, to achieve the 'pixelated look'.

The World Map was created by importing a set of freeware game tiles into a freeware level design tool called 'Tiled' (see appendix ??). This tool allows users to load a particular set of tiles and arrange them on a grid of their choosing. The tileset used for the World Map was originally created for the expansion and creation of new maps for the game "Tales of Middle Earth" but it lent itself very well to the task of creating a world map. After the first version of the map was completed, the location names and price tags were added with Photoshop CS 6 where other minor adjustments were also made.

### Items and Achievements

Most of the items and achievements icons were made by Alis, a pixel artist who made them available for public usage (see appendix ??). However a minor percentage of the items were created by modifying sprites of the popular game METAL SLUG by SNK/Playmore. These sprites are all properties of their respective owners and no copyright is claimed upon them.

# **Chapter 4**

## **Implementation**

### **4.1 Development Methods**

### **4.2 Heuristic Evaluation**

### **4.3 Technologies**

#### **4.3.1 Python**

Python is one of the most popular programming languages with great features such as a clear syntax and an extensive amount of documentation, libraries and native methods. The choice of using Python as a main programming language was very much tied to the web development framework chosen (Django). Nevertheless, Python has many advantages of its own, that became very important to the development of Gold Digger. Built in methods, such as *random.randint()* (to generate random integers) and *pow()* (for exponentiation) were of great help in the the development of the yield functions for each mine. Furthermore, thanks to the *list* object, it is very easy to create, manipulate, sort and modify both arrays and list, which proved very useful, given the structure of each mine. Finally, Python dictionaries, allow us to store pairs of values and retrieve them seamlessly in just a few lines of code which, given the number of variables each game object has to handle, is a great advantage, since then we don't need to handle handle them one by one.

#### **4.3.2 Django**

Django is a very powerful, well documented, easy to use, extensive web development framework based on python. In Gold Digger, it provides the base on which all of the other technologies employed are built upon. The choice of Django as a web development framework has been made for all the reasons listed above, but some were important in particular. Firstly, Django uses a Model View Template architecture. Each url request is handled by the Model middleware in Django and redirected to a specific View associated with it. The view, in turn, handles the request and provides the data to be passed to a the Template that further processes it and diSplays it (see below for Django template code). Another one of Django's main

features is the handling of SQLite databases through simple python commands. Because each user is associated with a User Profile object, frequent update of the database was needed in order to update all the game values and player stats. In Django, this is very easy and intuitive and accessing the database data can be done in any of the views seamlessly.

Furthermore, Django offers a template support through the use of **Django tags**. These Django-specific commands are enclosed in special `{% ... %}` tags and allow the advanced display of content and values sent from the views in HTML documents. Finally, Django offers session support, an easy login module, and a media server, which are all involved in the development and functioning of Gold Digger

### 4.3.3 HTML5

HTML5 is the most common markup language for presenting content on the web. All of the elements of each page (visible and not) are defined by HTML5 tags, creating a structure in which content can then be easily styled thanks to CSS3 stylesheets and manipulated using Django tags (see 4.3.2) AJAX calls (see 4.3.7) and JQuery. Two of the most important HTML5 attributes, are the **class** and **id** attributes. Classes allow similar elements to have the same style, thus allowing us to maintain integrity of graphics throughout the website without having to modify the parameters of every single element. Elements with the same classes will exhibit similar properties. Finally, the **id** attribute, allows us to refer to particular elements when we want to modify or alter them in some way using technologies such as AJAX and JQuery (see 4.3.6)

### 4.3.4 CSS3

As mentioned in the previous section, styling is applied to HTML5 elements via reference to CSS3 stylesheets. In CSS3 sheets we can define the parameters for the styling of each class, id or element that is present in one of the templates. If a template displays a link to a particular stylesheet, elements that display one of the classes defined in that stylesheet, will be formatted accordingly. In the example below, we can see how styling is applied to the element with id 'home':

```
1 #home{  
2     background-position: 25%;  
3     background-image: url(/ static / Wild%20West%20Background.png );  
4     background-repeat: no-repeat;  
5     background-size: cover;  
6     padding-bottom: 20%;  
7     padding-top: 5%;  
8     margin-bottom: 0;  
9 }
```

here, a background picture for the homepage heading is defined and set in place using the *padding* parameters. Padding is one of the parameters for positioning and styling elements of that are associated with the HTML box model.

Finally, CSS3 is indispensable in order to implement web page's responsiveness, in order for the pages to be displayed appropriately on screens of different sizes. There is more than one technique to make this possible but, in Gold Digger, we used **media tags** which specify the styling of particular items for different screen size ranges.

#### Animate.css

In addition to offering the possibility to style HTML5 elements, CSS3 also offers the possibility to apply animations to them. The Animate.css library (by Daniel T. Eden) in particular, makes this as easy as adding the 'animated' class to any element we wish to apply an animation to, together with an attribute defining the kind of animation. These animations are used mainly in the mines, to make each layer of scaffolding in the mine appear by descending from the top, as well as nicely presenting the new mine when it's first created. Since Gold Digger is fundamentally a video game, and since one of video games' fundamental features are their animations, it seemed non trivial to add some animated element to match user's expectations on this matter. Animate.css allows us to do this easily and cleanly.

### 4.3.5 Twitter Bootstrap CSS

Twitter Bootstrap uses a both CSS3 and Javascript to offer the possibility to apply sleek and minimalist themes to webpages that implement a link to them. The theme used for Gold Digger, is '**Simplex**' (by Thomas Park for Bootswatch) and it has been very useful for the organization and styling of the many elements of each page. Firstly, Bootstrap offers a very easy to use grid system, that allows us to easily divide any section of the page in columns and sub-columns. This can be seen throughout the website where the left side column hosts the side item panel, the middle column the main content (for example the layers of a mine) and the right column holds any additional information or just provides spacing. Secondly, Bootstrap comes equipped with Javascript plugins for the display of modals and tooltips. The former ones, for example, are used both in the world map to display the details of a location and on the main page, to host the registration form. Finally, Bootstrap allows to display continuity of colour and graphics throughout the website, since to equal elements always corresponds equal graphics.

### 4.3.6 Javascript and JQuery

JQuery is a very useful Javascript library that allows to apply special animations and features to elements of a webpage. By writing functions that are activated upon key press, click or hover, we can make webpages even more interesting and interactive than by simply using CSS3 and Bootstrap. There are two main applications of JQuery and Javascript that contributed to the look and functionality of Gold digger.

#### Trip.js

Trip.js is a very useful library to implement guided tours of an web app or website. It allows developers to design explanatory labels for each of the elements of a web page, with the option of highlighting them if necessary. Trip.js also incorporates Animate.css to allow the labels to be displayed with nice, tidy animations.

```

1 var trip2 = new Trip([
2   {
3     sel: $("#California_landscape"),
4     content: 'Hello_and_welcome_to_the_<b>Gold_Digger_tutorial!</b>',
5     position: "s"
6   },
7   ...
8 $(document).ready(function () {
9   trip2.start();
10 });

```

Here we can see one of the labels being defined and attached to the #California\_landscape element, with a particular message and position. Finally, upon page load, the `.start()` method is called on the Trip object (containing the array of labels) and the tour begins.

### **jquery-animate-numbers.js**

This very simple JQuery plugin allows to animate numbers so that all the numbers in a given interval are displayed in a 'rolling' fashion.

HTML

```
<div id="num">1234</div>
```

JS

```
$("#num").animateNumbers(4321);
```

In Gold Digger, this plugin is used in any page in which the amount of gold is incremented or decremented (like in a mine or in the shop). Animating numbers, like adding animations with Animate.css also contributes to the 'game feel' necessary for Gold Digger.

#### **4.3.7 AJAX**

AJAX stands for Asynchronous Javascript and it is an ensemble of technologies that, together, allow a web page to exchange data with the server without having to reload the page itself. This is achieved through an extra layer of interaction between the server and the client that is known as AJAX Engine. In Gold Digger, AJAX is essential, especially in the 'game pages' in order for the user not to have a disjointed gaming experience in which the page would have to be reloaded and be re-compiled every time. AJAX is also used in the general store page, to smoothly display the new item in the side item panel, and in the assignment of achievements, to display them as soon as they are earned. Thanks to AJAX we are able to keep the game experience close to what the user would expect while playing any other stand alone game. Since video games normally exist in a 'continuous environment', they do not have to deal with the limitation of separating content in different pages, however, with AJAX we are able to successfully compensate for this issue.

### **4.4 Development challenges**

Most of the development challenges we faced during the development of Gold Digger were related to the dual nature of Gold Digger. On the one hand, Gold Digger needed to be a game to experiment on searching behaviour and information retrieval, while on the other it had to be developed as a website, adding all the challenges of web development to the development of a game. Video games, even simple ones, usually exist in their own environment, in which all the variables needed can be easily accessed and modified. In a website instead, this continuity needs to be compensated for with session variables, access to user objects and above all, interaction with the database.

Other challenges in the development of Gold Digger, were the design of the game mechanics and the implementations of the methods and functions involved in the creation of the yield and cue arrays. These last ones needed to produce values that, while being generated by a function, would still appear random enough to the user so as to offer a more challenging and interesting gaming experience. In the following subsections we outline some of the strategies used to overcome these challenges.

#### 4.4.1 Creating mines

(1) By selecting a location from the world map, the user sets the session variable 'location' to the name of the location selected and the '**has\_mine**' parameter to an empty string. Then, the request is redirected to the **game2** view in order to initialise all the game variables

(2) The **game2** view first checks if the '**has\_mine**' variable contains an empty string (meaning that this is the first mine of the day) and then uses the 'location' previously set (see (1)) value to determine which yield generator to select to generate yields. There are six different yield generators. Each one will generate a different array of ten values, representing the amount of gold in each layer of the mine.

(3) Once we have a YieldGenerator, we need two more values to generate a mine. These are: '**accuracy**' and '**user**'. The former one contains the value of the tool's modifier which will be needed when generating cues. The latter one is needed for logging statements. By creating a **Mine** object, we automatically trigger the *make\_mine()* method in the **Mine** class which, in turn, triggers the **make\_yields** method of the YieldGenerator thereby starting the process of generating yields.

(4) All the different yield generators inherit the **make\_yields** method from the YieldGenerator class and define their own. This way we are able to add as many yield generators as needed, by simply adding a new method implementing **make\_yields**. Each yield generator defines its own static function that will generate an array of yields according to a specific equation that also introduces some randomness. Below, we can see the static function in charge of determining the yield of mines in Scotland.

```
1 def scoti_quadratic_function(x, a, k):
2
3     b = random.randint(-10, 5)
4
5     y = (a*(pow((x - k), 2)) + 90)+b
6
7     if y < 0:
8         y = 0
9
10    rounded = int(round(y))
11    return rounded
```

On line 5 we have a quadratic function generating a parabola which it takes **x**, **a**, **k** and **b** as parameters. The value of **x** goes from 0 to 9 and it represents each of the layers of the mine. **a** determines the steepness of the parabola and **k** determines the position of the parabola on the x axis (see A for a graph representation of each mine). This function will return a value **y** for each of the values of **x**. Each of these values, in turn, is rounded to the nearest integer and added to the array of yields. Finally, as we can see on line 3 a certain degree of randomness is introduced with **b**, which modifies the height of the parabola and consequently each one of the yields. **k** is also a random number in a range, this way, we don't run the risk of producing the same mine over and over again. All of the static functions in each of the YieldGenerators has some degree of randomness.

(5) A cue is a number from 0 to 6 determinig which of the gold flecks patterns will be displayed over an undug layer of the mine. As we will see in the next section, each of these images is associated with a class and Django tags will use the value of 'cue' in each block to display the appropriate one. Once the array of yields is ready and has been received by *make\_mine()*, it is passed to **cuegen.py** together with the value containing the highest of the yields and the 'accuracy' level we discussed in (3) in order to begin the process of determining the cue corresponding to each yield. In **cuegen.py** we set the number of possible cue patterns, then, we go through the following operations:

- Determine the cue range: this is the maximum yield divided by the number of cue patterns. A different cue (gold flecks image) will be displayed in association with each range.
  - Determine the span: this is the span in which the cue will be chosen, the wider the span, the least likely the cue displayed will match the yield. It is a function of the maximum amount of point and the accuracy of the scanning tools. In the table below, we can see an example in which the maximum yield of the yield\_array is 42. If for instance the yield is x and the user's scanning equipment is the lamp the cue span will be wider (orange) and any of cue the cue patterns 1 to 3 might be displayed. However, if the user is using a sonar (green), which is more accurate, the cue pattern displayed will be either 2 or 3. This way, we are able to display cues in relation to the accuracy of a user's item. The better the item, the smaller the span.



Figure 4.1

- Choosing the cue: as discussed above, a value in the range of the cue span is chosen at random and the associated cue appended to the array of cues which is then returned to the Mine class

(6) Now that we have both the array of yield and the array of cues, we are able to assign values to each of the **Block** objects representing each a layer of the mine. Each Block object, in fact, contains the following values:

- **pos**: the position of the mine, an integer from 0 to 9
  - **gold**: the yield
  - **cue**: one of the 6 cue values representing one of the patterns
  - **dug**: a boolean determining if the mine has been dug or not

(7) The array of blocks is sent over from the *game2* view to the template, in a dictionary, together with all the other values relevant to the display of the game page.

#### 4.4.2 Digging: sessions AJAX

A fundamental functionality that needed to be implemented in Gold Digger was the ability of the user to continue digging into a mine without having the page reload. When digging, a certain amount of gold is acquired, while a certain amount of time units are spent. At the same time, gold flecks layers or shaded layers of scaffolding, need to be replaced with the appropriate graphics and animations. Django helps us very much in the display of all the items needed in this process, however, once the page is rendered and displayed, we would need to reload it for any content to be altered, by redirecting to the appropriate view and get back to the template.

The first step in addressing this problem has been the implementation of sessions. Sessions support comes pre-built in Django and it is thus very easy to take advantage of. Sessions are employed mainly in maintaining each mine constant upon digging or reloading the page. Each mine, once created and until the moment when the user decides to move away from it,

needs to remain constant, this means that its gold yields would not be generated anew and that, furthermore, the user will be able to see the amount of gold harvested from the layers she already dug.

To achieve this, we make use of two main session variables: '**pointer**' and '**pickled\_blocks**'. The '**pointer**' variable is responsible for holding the value of the position of the right hand side buttons used to dig and move while the '**pickled\_blocks**' variable is responsible for holding the pickled array of blocks representing the current layers of the mine. Because there are several passages involved in this mechanic it is probably helpful to break them down into steps:

- (1) When the user enters a mine, the **game2** view is accessed. This view is responsible for returning a dictionary to the game template with all the values relevant to the mine and the player's status. The '**pointer**' variable is set to 0, while the array of blocks representing the layers of the mine is 'pickled' (see 3.11b) in order to be added to the session values and used later.
- (2) The template receives the dictionary sent by the **game2** view and displays it with the help of Django tags. First of all, we check if the dictionary contains an array of blocks, if it does we proceed to further conditional statements to decide how each one of those blocks is going to be displayed (see below).
- (3) First of all, we check if the block has been dug or not. This is done by checking if the '**dug**' parameter is True or False. Depending on the case, different graphics will be displayed.

```

1  {% if blocks %}
2  <div class="col-xs-5_mine">
3  {% for block in blocks %}
4  {% if visibility > block.pos %}
5  {% if block.dug == False %}
6  <div class="row_flecks_{{block.cue}}_animated" id="goldlayer_{{block.pos}}">
7  <div class="scaffold_1_shaded" id="scaffoldlayer_{{block.pos}}">
8  <div class="points" id="points_{{block.pos}}"></div>
9  <div class="comment" id="comment_{{block.pos}}"></div>
10 </div>
11 </div>
12 {% else %}
```

In the code snippet above, we can see what happens if the '**dug**' variable of a block is False. Thanks to a particular kind of Django tags (the double curly braces) we can display the appropriate cue, granted that our visibility (line 4) is greater than the number of the layer we are in at the moment. On line 6 we can see how *class=row\_flecks\_{{block.cue}}* will display a different row of flecks according to the value of *{{block.cue}}*. A similar process will take place if the '**dug**' value of a block is True, except in this case we will use *class=row\_nuggets\_{{block.cue}}*, which displays the appropriate amount of nuggets and the scaffolding of an already dug layer.

(4) Now that all the layers of the mine have been correctly displayed, we need to take care of the right side panel which hosts the game controls: the **dig** and **move** buttons. To each layer of the mine corresponds a row containing a button couple on the column to its right. However, all of them are hidden from the user except the one corresponding to the the first undug layer. Once the user clicks the **dig** button, the row is removed and the '**hidden**' attribute is removed from the row below it so that the button can be shown next to the next layer. At the same time, the amount of gold dug in the previous layer is displayed.

(5) This modification of the DOM elements properties is achieved through AJAX and JQuery. When the user clicks on the **dig** button (or Enter is pressed), an AJAX event is triggered. The number of the layer and the amount of gold that it contains are sent to **dig-ajax.js** together with the *csrf middleware token* (required by Django for security purposes) so that they can be used in the **ajaxview** view where all necessary updates to both session and database variables will be made.

```

1     request.session['pointer'] += 1
2     request.session['time_remaining'] -= user.tool.time_modifier
3     request.session['gold'] += int(round(gold_dug*user.tool.modifier))

```

Here, we can see how the session variable 'pointer' is incremented by , while 'time\_remaining' and 'gold' are updated in relation to the items possessed by the user.

(6) In **ajaxview**, we also take care of un-pickling the array of blocks held in one of the session variables and update it. This is done in order to be able to maintain the array of blocks constituting the current mine until the user moves to a different one. If this didn't happen, the user would be able to simply reload the page and get a new mine every time at no cost. For this reason, it is very important that the array of blocks is maintained until a new mine is generated, otherwise, users who took advantage of this glitch would be able to gain an unfair advantage and the data extracted would not be reliable.

```

1 # Unpickling the blocks
2 pickled_blocks = request.session['pickle']
3 blocks = pickle.loads(pickled_blocks)
4
5 # Updating the block values
6 blocks[pos].dug = True
7 blocks[pos].gold_extracted = gold_extracted
8
9 # Pickling
10 pickled_blocks = pickle.dumps(blocks)
11 request.session['pickle'] = pickled_blocks

```

(7) At this point, a JSON object (*myResponse* containing all the necessary values is compiled and sent back to **dig-ajax.js** where, thanks to JQuery, we can make the appropriate modifications to update the game page.

```

1 myResponse['totalgold'] = user.gold
2 myResponse['timeremaining'] = request.session['time_remaining']
3 myResponse['currentgold'] = request.session['gold']
4 myResponse['goldextracted'] = gold_extracted
5
6 return HttpResponse(json.dumps(myResponse), content_type="application/json")

```

(8) In **dig-ajax.js** we use a combination of JQuery methods and plugins in order to update the content. In the code snippet below, we can see how the elements with id 'totalgold' and 'currentgold' as well as the 'progress-bar' class are updated. For the first two (lines 2 and 4) we use the **.animateNumbers** (see 4.3.6) plugin in order to make the digits 'scroll' in videogame-like way, while, for the last one (line 3) we simply use the JQuery method **.css** to change the **width** attribute of the element, thus shortening the bar indicating the remaining amount of time units. Finally, in lines 6 to 8, we make sure that the scrollbar is positioned in line with the first undug layer, so that the user doesn't have to scroll down the page every time she finished digging the visible layers.

```

1 // Update points
2 $('#totalgold').animateNumbers(response['totalgold'], true, 200, "linear");
3 $('.progress-bar').css("width", response['timeremaining']+ "%").html(response['timeremaining']);
4 $('#currentgold').animateNumbers(response['currentgold'], true, 200, "linear");
5
6 if ((pointer-1)>0){
7     $('html, body').animate({scrollTop: $('#scaffoldlayer_'+(pointer-1)).offset().top}, 500);
8 }

```

In a similar way, we also update the mine layers to display the appropriate amount of gold nuggets and the right scaffolding graphics. As discussed in (4), here we also take care of revealing the next layer of buttons by removing the 'hidden' class from it, while at the same time replacing the previous layer's buttons with the amount of gold dug.

(9) In the game template, the current gold and time remaining value are updated, the gold nuggets displayed and the scaffolding brought in with an animation. If the user now decides to change mine and move, a new mine is created in the *game2* view and the page is reloaded. Since in this case the digging needs to start from the top again.

## 4.5 Testing

### 4.5.1 Unit Testing

There are several key areas in the code for Gold Digger, as we've seen in section 4.4 there are various functions that not only are fundamental to the functioning of the game, but that can modify its behaviour without us noticing. For this reason we've implemented a series of unit tests. This way, at each stage in development, we can run the test and check that everything still works according to plan. Unit testing is another very useful Django features which allows us to simply write tests in *tests.py* and launch them all at the same time from the terminal. The unit tests for Gold Digger are divided in four classes, each comprising of different tests:

- **ModelTests:** this class include tests relating to Gold Digger's models.
  - **test\_user\_profile\_data:** checks if a user can be successfully added. The test fails if any of the parameters of the UserProfile model don't correspond to expectation.
- **FormTests:** this class include tests relating to Gold Digger's forms.
  - **test\_clean\_password:** checks to make sure that a user who enters a confirmation password that is different from her first password fails authentication.
- **CueTests:** includes test related to the appropriate generation of cue values
  - **test\_appropriate\_cue:** checks that, given an array, a maximum of gold and a scan value, *tcuegen.py* produces the appropriate cues.
  - **test\_cue\_function:** checks that, given an array, a maximum of gold and a scan value, *cuegen.cuefunction()* produces the appropriate span values (see 4.4.1).
- **GameTests:** includes test related to the appropriate functioning of yield functions. The first six tests were developed in the first stages of the development of Gold Digger. They test six YieldGenerators that are not used in the present version. However, we made the decision to keep them in case future development demands for the use (and testing) of those YieldGenerators. The following tests were developed in order to test the presently used YieldGenerators as well as the creation of a mine.
  - **test\_constant\_yield**
  - **test\_random\_yield**
  - **test\_linear\_yield**
  - **test\_cubic\_yield**
  - **test\_exponential\_yield**
  - **test\_quadratic\_yield**
  - **test\_positive:** because the YieldGenerators used in the most recent iteration of Gold Digger generate yield according to functions which are somewhat randomized, it is difficult to test their output. Instead, here we check if any of the yields is negative, since we don't want the player to dig negative amounts of gold.

- **test\_mine**: this lasts test checks if Mine objects are appropriately created. The test fails if any of the 'cue' or 'gold' values in each block (see 3.4) is equivalent to *None* or if any of the position values are not comprised between 0 and 9.

All the above tests are successfully passed in 2.625 seconds.

#### 4.5.2 Live Testing

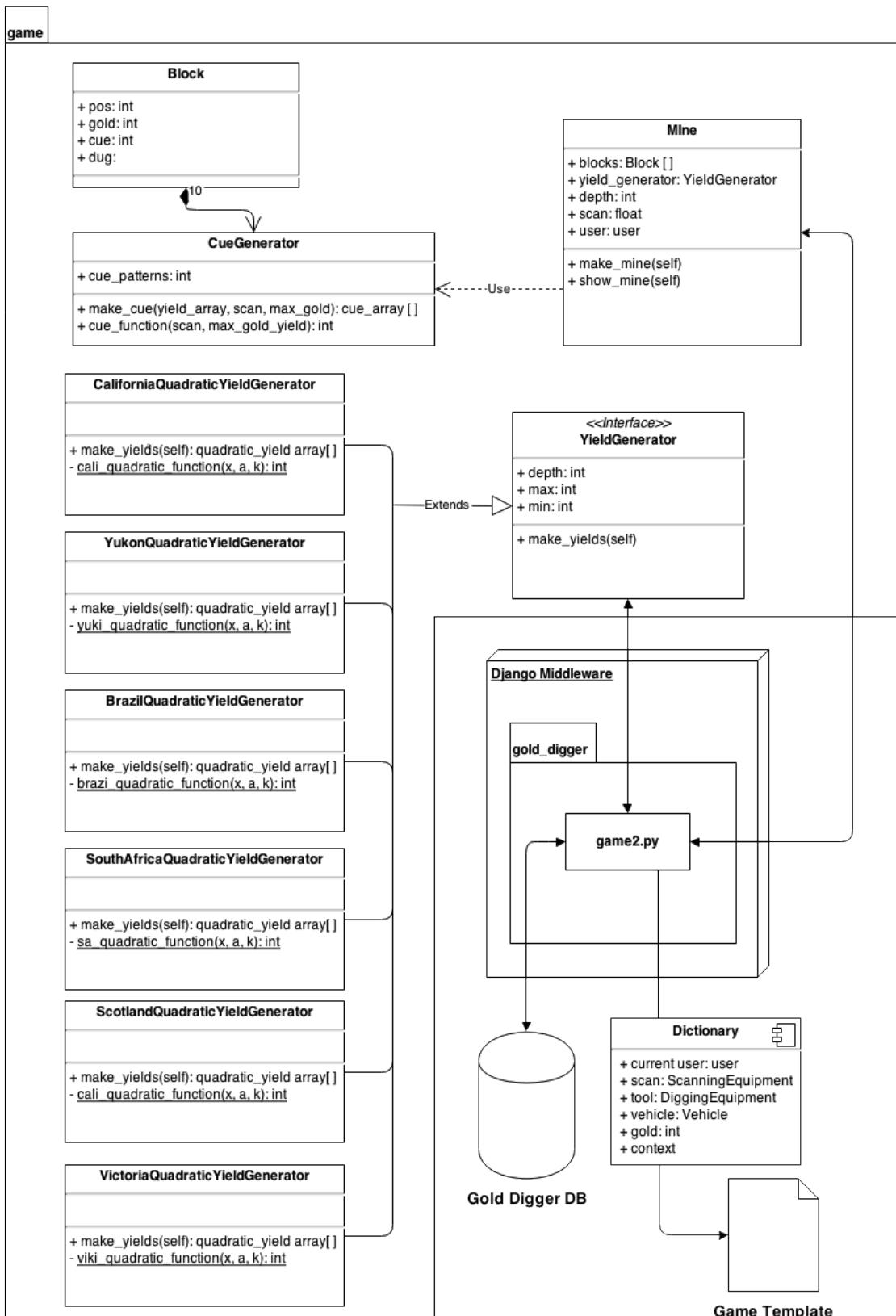


Figure 4.2: The class diagram illustrates the way the game object is prepared and sent to the template for display

## **Chapter 5**

# **Results and Data**

**5.1 Data Logged**

**5.2 Data Analysis**

**5.3 Results**

**5.4 Reflection**

## **Appendix A**

# **First appendix**

### **A.1 Section of first appendix**

## **Appendix B**

### **Second appendix**

# **Bibliography**