CBL TheHolyBeir

TheHolyBeir is a game created by Kornél and Javier. It's a one-chance click game in which you advance through levels in order to get as far as possible. There is also a bonus system which increases as you kill more bosses, yet keep in mind acquiring heals will lower this score. Every 25 levels there is an unskippable main boss and a final boss every 100 levels (main bosses (:|) have double the stats of a normal boss at that level, and final bosses(}:() have 3 times the stats).

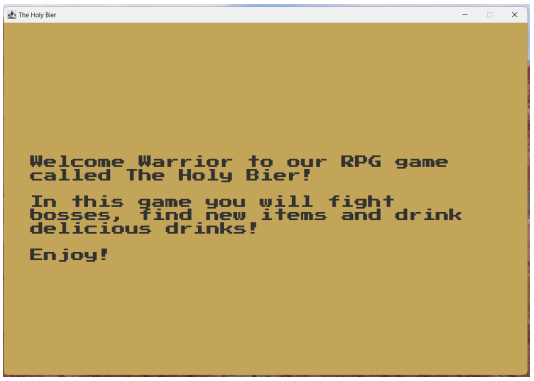
Earlygame: lvl1 to lvl 75
Midgame: lvl 75 to 100
Endgame: lvl100(excluded) onwards

Note that a run is considered successful once level 100 is beaten, and endgame is just there to challenge players who want to continue their experience.

Gameplay Explanation:

Tutorial:

Intro screen:

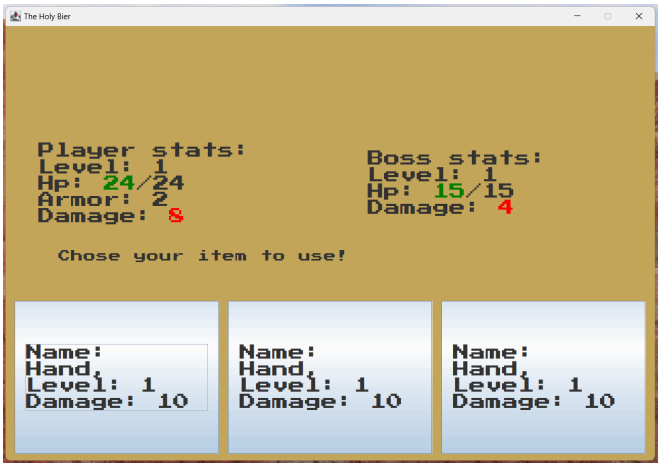


When you first start a run, the main screen will appear:

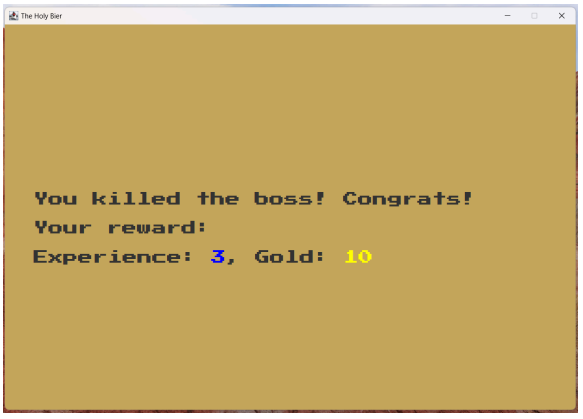
The player is immediately shown the main part of the game. There are three rooms: item room, where you can acquire items; boss room, where you fight bosses; and the Pub, which the player doesn't know what he can do in. The player has two options, either entering the boss room or the Pub. The player chooses the boss room.

The main fight screen will then appear:

There are three buttons: heal for healing, fight for fighting and escape to escape.
The player clicks fight since his health is full.
A screen with different options to attack will appear.

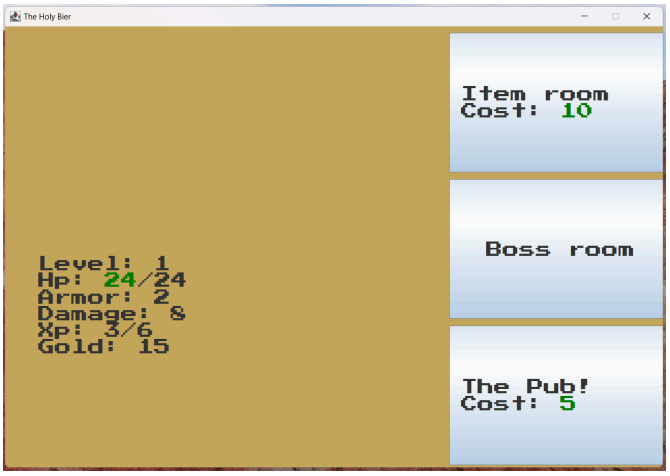


From this screen, the player can see how there are three possible weapons, and chooses one to atack.
After attacking, he will see how his hp lowers, and with some simple math he can realize that the damage he received was the damage of the boss minus his own defense, same goes for is own damage (as if not he wouldn't have been able to kill the boss in one atack).



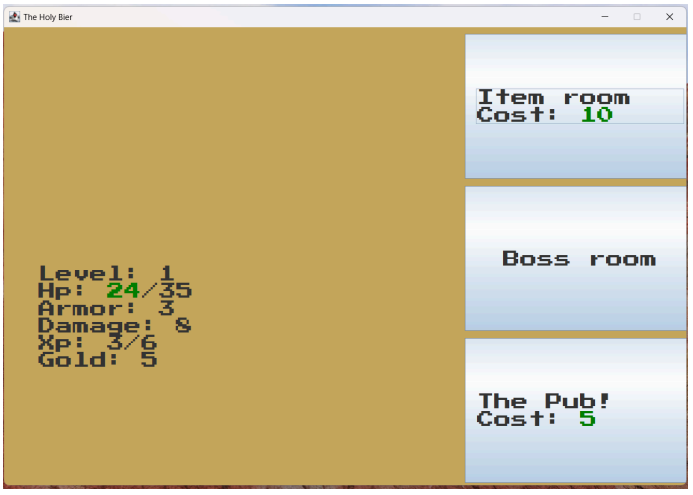
The fight is over and a screen with the player rewards will be prompted.

The player then returns to the main screen.



With enough money he'll either buy an item or go to the pub. He decides to buy an item.
A screen with a new item and the inventory of the player is prompted.

Once the player clicks one of will return to the main screen:



With some money left, the player will click the pub and a new screen will be prompted.

The player can see how he can store a maximum amount of 3 heals, clicks one of the buttons and returns to the main screen.

Now the player has seen the basics of the game.

This is one example, yet the player will learn all this basic in the first minute of gameplay in a similar way (order of things might differ from each other) since the player doesn't have that many options due to money at the start, given the player plays normally.

As the player continues playing, he will discover how the bosses level up each time he gets to the main screen (same for items), meaning he cant abuse the pub nor pick items recklessly, and discover that the price of the pub and the amount of health it heals each time is random (these are level 0 in order to show the player they don't level up). He will also discover that every 25 levels there are harder, unskippable bosses, with the hardest being those every 100 levels…, essentially the player will learn how to play the game without the need of an explicit tutorial.

The point of all of this was to make a tutorial for the player without the player realizing it is a tutorial (in other words, making a good tutorial. We limit what the player can do at the start of the game to guide him to discover the things we want him to learn.

(How to make a good video game tutorial - SCHOOL OF GAME DESIGN, How to design a perfect game tutorial? - Try Evidence, the websites have more info. than the one needed in order to create the tutorial for our game due to its simplicity. Also inspired by other game's "tutorials" such as Mario Bros or Dark Souls)

Breef gameplay explanation:

The player advances floors. In each floor he can choose to buy an item (item shop), fight a boss or buy a heal (The Pub). The objective of the player is to reach the highest floor possible, defeating as many bosses as possible, specially final and main bosses (every 25 and every 100 levels respectively).
The combat system is simple, turn-based where the objective is to defeat the boss you are currently fighting. The player can choose either to heal, escape or attack the boss with any of the three weapons in his inventory. If the player decides to escape, the boss will damage him before he manages to escape. When the player dies, the run is over. Once the player dies, a bonus screen will be displayed, which will have the amount of bosses killed ;and, items and heals bought by the player. Bosses killed add 13 bonus points per boss while heals subtract 3 bonus points per heal to your final score. In this way, the player is incentivized to kill as many bosses as possible and use as little healing items as necessary.

STATS, BALANCE AND MECHANICS:

Items + Player:

For the development of the game, we opted for a combat system based around items and the player. Items work in a way that they are complementary to the player, meaning both items and player are equally important to level up; therefore, the player has to have in mind he has to fight as many bosses as possible to level up.

In order to increase the game's variety and freshness, we included several core game aspects:
(Game Design: How to make players care | by Natalie | Medium)

Random aspect:

Stats of the items vary each level in order so the player doesn't see the same numbers all the time, to avoid the game feeling repetitive. There are also rarities and types of items in order to make the game feel more unpredictable and add variety to the gameplay. (The Significance of Randomness in Games | by OvalPixel | Medium). Same goes for the bosses and the heals (though these have no rarities). The idea of all this is increasing the game's replayability.

Variety of items:

For the game we created 4 different types of items: swords, axes, grenades and spears, in order to give the player a sense of choice. Each of these weapons is unique in its own way.

Swords are a balanced item, where every stat is and grows equally fast.
Axes are a more damage focused weapon, excelling at their damage and Hp but have lower armor.
Grenades are glass-cannon items, where they have the best damage out of all items but have absurdly low hp and armor, so in later stages of the game you either kill or get killed.
Spears are the defensive option, where they have a lower than average atack but a high armor and hp.

We wanted to add classes to the game like those in RPGs (bandit, warrior..), adapting this concept to our game idea, reason why each type scales differently and has different stats in the first place. (Book: How to make an RPG: By Daniel Schuller RPG Stats: Implementing Character Stats in Video Games (howtomakeanrpg.com), How to Make an RPG - DOKUMEN.PUB))

Cycling of items:

The player can have a maximum of 3 items in their item inventory, yet must change one of these each time he receives a new one. As the level of the item increases each time you reach the main screen again, this isn't something that will negatively influence the player's run, and will just force the player to change his approach depending on the type of items he gets (specially in levels close to the final boss [75 - 100] and the final boss itself). This is essentially another way of making the game experience more unpredictable to contribute to the game's replayability (The Significance of Randomness in Games | by OvalPixel | Medium).

Formulas used for stat scaling of items: (see program for the different values each of the randNum can take)

Sword:

Hp: $(e^{0.04*level} - 1) * 10^4 * randNum / 7500$
Armor: $Hp * 0.1$
Damage $(e^{0.04*level} - 1) * 10^4 * 0.5 * randNum / 7500$

Axe:

Hp: $((e + 0.01e)^{0.04*level} - 1) * 10^4 * randNum / 7500$
Armor: $Hp * 0.075$
Damage: $((e + 0.01e)^{0.04*level} - 1) * 10^4 * 0.5 * randNum / 7500$

Grenade:

Hp: $((e - 0.08e)^{0.04*level} - 1) * 10^4 * randNum / 7500$
Armor: $Hp * 0.025$
Damage $((e + 0.04e)^{0.04*level} - 1) * 10^4 * 0.5 * randNum / 7500$

Spear:

Hp: $((e + 0.01e)^{0.04*level} - 1) * 10^4 * randNum / 7500$
Armor: $Hp * 0.15$
Damage: $((e - 0,02e)^{0.04*level} - 1) * 10^4 * 0.5 * randNum / 7500$

The main formula used was $(e^{0.04*level} - 1) * 10^4 * randNum / 7500$, used for the sword as its a balanced weapon, the rest of the formulas were calculated after this one. At the same time, this formula derives from the formula used to calculate enemy stats, which was made along with the formula of leveling up in order that there is a point in which the rate at which the player levels up slows down as the game progresses.

We opted for using an exponential growth system when increasing the stats of items, weapons and bosses. This was mainly because a bigger difference between numbers and along with bigger numbers in general give the player a larger stimulus and a sense that they are being rewarded better, to keep the player playing. (The Psychology Of Games: The Unit Effect And Player Perceptions, Roguelike games: The way we play (researchgate.net), Game Design: How to make players care | by Natalie | Medium).

Yet, in order to calculate damage to the player, we opted for a flat reduction of damage (damage - armor) (unlike what Damage formulas (yujiri.xyz) recommends), since we wanted to keep the game as simple as possible so the player can get used to it quickly and avoid them losing their interest on our game due to too complex mechanics (What Is Games 'User Experience' (UX) and How Does It Help? | by Player Research | Medium). We did use an exponential system to increase these numbers, similar to how the first mentioned article recommends.

By keeping the system like this, the player can easily tell how much damage the enemy will do and make a plan accordingly. This in order to help the player while he is learning the game so he can easily adapt before its difficulty increases (Digital Game Dynamics Preferences and Player Types: PREFERENCES IN GAME DYNAMICS (researchgate.net)). We want the player to think he is *defeating enemies* rather than *dealing damage to enemies*, so that the player finds his gameplay more appealing (Digital Game Dynamics Preferences and Player Types: PREFERENCES IN GAME DYNAMICS (researchgate.net), Roguelike games: The way we play (researchgate.net)).

The player is essentially an item which can level up (Main aspect which isn't in some way random to make sure player progression feels consistent every run).
Player:

Hp: $((e + 0.01e)^{0.04*(level+1)} - 1) * 10^4 * 2.5 / 100$
Armor: $Hp * 0.05$
Damage: $(e^{0.04*level} - 1) * 10^4 / 100$

Item rarities:

There are 4 rarities: tier 1(75%), tier 2(20%), tier 3(4%) and unique(1%). The first three are straightforward, each has 0, 10 and 20 extra levels respectively (ex. A tier 2 weapon obtained in floor 10 will have lvl 20). On the other hand, unique tier has 30 extra levels and depending on its typing a unique name and unique ability (life steal (sword), double damage (axe), damage nullification (spear) and oneshot (grenade)).

lifesteal: heals 25% of the damage dealt when using the unique sword.

double damage: 80% chance of dealing double the damage when using the unique axe.

Oneshot: 20% chance of killing the enemy in one atack when using the unique grenade.

Damage nullification (passive): 50% chance of nullifying the enemies attack whenever having the spear in the inventory.

ITEM COMMENTS OF FUNCTIONS:

rarityCalc(): calculates the rarity of each of the items to be created, extraLevel is the amount of extra levels the item will have when created.
Public Sword(): defines a sword item by calculating its stats corresponding to its rarity and the room level and, if unique, is given its passive (lifeSteal = true).
Public Axe(): defines an axe item by calculating its stats corresponding to its rarity and the room level and, if unique, is given its passive (doubleDamage = true).
Public Grenade(): defines a grenade item by calculating its stats corresponding to its rarity and the room level and, if unique, is given its passive (oneShot = true).
Public Spear(): defines a spear item by calculating its stats corresponding to its rarity and the room level and, if unique, is given its passive (damageNull = true).

PLAYER COMMENTS (program):

variable **maxHp:** it's the total hp of the player + its items. (player hp is used when increasing the stats of the player when leveling up).
variable **maxArmor:** it's the total armor of the player + its items. (player armor is used when increasing the stats of the player when leveling up).
NOTE: Each time an item is changed or the player levels up, these have to be recalculated.

getCurrentDamage(): calculates the damage the player does by taking the damage of the item chosen to atack, as well as implements unique passives (except damage null which is done in controller due to access necessities for the program) and adds a 10% critical chance.
Public Player: defines a player with an empty inventory of items (filled with hands) and heals (filled with empty bottles), along with its initial stats, its initial level (1) and the amount of xp the player starts with (0) alongside the amount of xp needed for the next level.
playerReward(): increases the gold of the player by the amount of gold a boss gives when defeated, and increases level according to the xp gained by killing the enemy. It works in such a way that the player can level up several levels if the amount of xp earned allows this (essentially it works in the same way as any normal level up system such as pokemon).

Bosses:

Bosses are the enemies in our game. Every 25 levels there is slightly stronger main boss (*2 stats) and every 100 levels a final boss (*3 stats).

Boss (randNum [200 - 210]):

Hp: $(e^{0.04*level} - 1) * 10^4 * randNum * 1.75/10000$

Damage: $(e^{0.04*level} - 1) * 10^4 * randNum / 25000$

By having stronger enemies every few levels (specially final bosses), the player gains a new objective to look forward too, keeping him engaged and in this way creating spikes of difficulty in the game (Digital Game Dynamics Preferences and Player Types: PREFERENCES IN GAME DYNAMICS (researchgate.net)).

The objective of normal bosses is to level up the player so he can face the stronger bosses ahead. All bosses give 10 gold and the amount of Xp they give is equivalent to a third of the mean of their stats.
By making the amount of xp given by a boss linked to their average stats, it improves the rewarding system as the more difficult (higher stats) a boss is, the more xp it will give.

The reason for the randomness present is the same as that of the one in the items.

BOSS COMMENTS (program):
dealDamageTo(): takes the damage of the boss and the armor of the player, subtracts the armor to his damage, and returns the new damage, which will be the damage that the boss deals to the player.
Public Boss(): defines a boss with random health and damage, according to the current floor level. Every 25 levels a main boss is generated instead(*2 stats) and every 100 a final boss(*3 stats). The xp and gold they are worth is double the normal boss amount (to increase the reward they give even more as these are harder fights, so the player feels rewarded and there is a slight decrease in difficulty after the battle to leave time for the player to recover).

Heals:

The healing system is a simple but effective system where they heal a percent of the player's max Hp. These can heal from 10 - 100%, increasing in 10s (10,20…,100) and cost between 1 and 5 gold, both of which are random. The player can have up to 3 heals in their inventory and is forced to change one of them each time he gains a new one. In this way, we make the player decide if he really needs a new heal. These can be obtained in the pub.

Reason why we went for a percentage increase of the max Hp was due to balancing issues, as if not we would have to balance another mechanic in the game. Since heals are used constantly during the run (specially on unlucky ones), if heals also had levels as the rest of items the game would become exponentially more difficult, which we wanted to avoid. They have level 0 so the player knows these don't level up. (Book: How to make an RPG: By Daniel Schuller RPG Stats: Implementing Character Stats in Video Games (howtomakeanrpg.com) , How to Make an RPG - DOKUMEN.PUB)

The reason for the randomness present is the same as that of the one in the items. In a similar way, heals were included in the game as it's a common feature in RPGs, as it expands the player options.

Overhealth: whenever grabbing an item lowers your maximum hp, the player will keep any amount of health he had beforehand, even if this means its health will be higher than the maximum hp. This is done in order of minimizing the negative effect of having your maximum hp lowered.

HEAL COMMENTS (program):
variable **healPercentage:** percentage of hp that the heal heals.
getHealAmount(): the actual amount of health the heal heals when used.
Public EmptyBottle(): defines a heal which heals 0% of max hp. Whenever a heal is used by teplayer, it is replaced by this object.
Public Beer(): defines a heal with a random beer name which heals a random percentage of the player maximum hp (10% - 100% in intervals of 10%) and with a random gold price (1 - 5 gold).

Gold and Xp:

Formula for calculating the amount of xp needed for the next level of the player:

$(e^{0.045*level} - 1) * 10^4 * randNum * 1.2/100$

The idea of the formula is that the higher the level is, the harder it is to level up since it grows faster than the boss stats. As mentioned before, xp is used as a tool for rewarding the player for killing bosses, to a greater extent than the bonus screen.

On the other hand, the amount of gold awarded to the player stays constant along the whole run (except in main and final bosses which give double the amount). This is off course done on purpose, so the player has to manage its resources during the whole run, something which most players find appealing in games (Digital Game Dynamics Preferences and Player Types: PREFERENCES IN GAME DYNAMICS (researchgate.net)).
It might look as if money is abundant throughout the game, but it isn't. Once the player starts approaching level 100, he will need to buy several items in order so it can have the best odds of winning against the final boss (the main strategy is to accumulate gold for about 75 levels, and afterwards spend it all on items till reaching the final boss every 100 levels.). It is also used as a tool to limit what the player can do at the very start of the game.

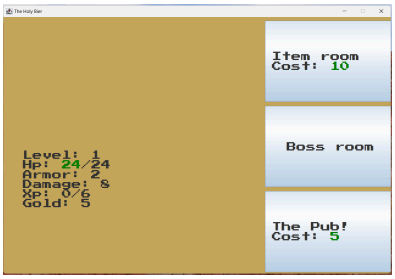
Balancing notes:

Weapons are balanced in general, with swords and grenades being good in early-game but getting worse in late game, while lances and axes are better for late game, as axe is the most consistent weapon to deal damage in these stages (due to its better defensive stats than the grenade).
Grenades have so much damage on purpose, as if it's not able to kill the enemy in one or two blows, it's probable the player will lose the run, especially in the late-game.
Any tier 2 weapon and above is supposed to be overwhelmingly superior to enemies and items in your current level, but eventually fall off due to how their stat calculation works (reason for their probability).
Unique weapon's passives are supposed to be strong, specially the damageNull, as it is necessary in late end game (200+) in order to increase your chances of winning against final bosses at these stages due to the enemy's exponential growth.
Balance changes were done with feedback of playtesters (Microsoft Word - REPORT_Metodologias_Catalunya_engOK.docx (gencat.cat)).

GUI:

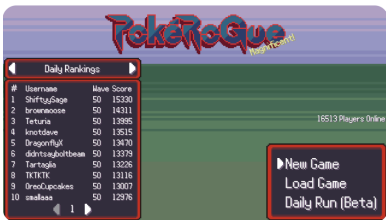
In order to make the GUI of our game, we made sure that it was simple and clear to follow, in order to avoid the player feeling overwhelmed by the amount of information displayed at once. Furthermore, we highlighted important stats and information for the player, to grab their attention and ensure they are playing the game as intended without missing anything(Designing a menu in computer games | by Alex Stargame | Medium, How to Design Game Menus and Interfaces for UX (linkedin.com)). We based the GUI on some examples to make sure it was appealing enough.

In the mainScreen, we made sure that the pub is at the bottom of the interface, to dissuade players from using it excessively, and incited the idea of acquiring new items and defeating bosses by placing these higher in the interface (The Science Behind Game UI/UX Design: Understanding Player Behavior and Interactions (designyourway.net)).



Examples:

What they all have in common is that they are simple and easy to understand, without flash graphics or colors / animations.



SOURCES:

(An example with everything and more than needed)
Book: How to make an RPG: By Daniel Schuller
RPG Stats: Implementing Character Stats in Video Games (howtomakeanrpg.com) ,
How to Make an RPG - DOKUMEN.PUB
(157 -158 [PANELS]), 199 - 210 [ATTACK CHOICES],
(215 - 219 FITTED TEXTBOXES), (220 - 222 [PROGRESS BAR]),
(259 - 274 [MENUS]), 275 - 282 [ITEMS],
(194 - 197 [MAIN MENU]), 403 - 416 [STATS], 417 - 425 [Levels],
431- 434[Stat Growth], 435 - 481[Might want to read for GUI], 480 - 488[Equipment],
*, 528 - 538 [COMBAT QUEUE], 562- 613 [Combat States], 788 - 792 [Sensible Stats]
What Is Games 'User Experience' (UX) and How Does It Help? | by Player Research | Medium

How to design a perfect game tutorial? - Try Evidence
How to make a good video game tutorial - SCHOOL OF GAME DESIGN

Microsoft Word - REPORT_Metodologias_Catalunya_ engOK.docx (gencat.cat)
The Psychology Of Games: The Unit Effect And Player Perceptions (gamedeveloper.com)
Roguelike Games - The way we play: International Journal of Engineering and Management Sciences (IJEMS) Vol. 7. (2022). No. 4
Roguelike games: The way we play (researchgate.net)

Damage formulas (yujiri.xyz)
Digital Game Dynamics Preferences and Player Types: PREFERENCES IN GAME DYNAMICS (researchgate.net)
Microsoft Word - REPORT_Metodologias_Catalunya_ engOK.docx (gencat.cat)
Game Design: How to make players care | by Natalie | Medium
The Significance of Randomness in Games | by OvalPixel | Medium

[Designing a menu in computer games | by Alex Stargame | Medium]
[How to Design Game Menus and Interfaces for UX (linkedin.com)]
[The Science Behind Game UI/UX Design: Understanding Player Behavior and Interactions (designyourway.net)]

Public opinion: *(not really used, more to see if comments in line with other websites)*

[Graphics - Tips To A Good Main Menu? | GameMaker Community]

Examples:

What they all have in common is that they are simple and easy to understand, without flash graphics or colors / animations.