International Journal of Advances in Engineering and Management (IJAEM)
Volume 5, Issue 6 June 2023, pp: 62-69 www.ijaem.net ISSN: 2395-5252

Greedy Based Improvised Cookie Clicker Bot

Team Lead- Pratyush Arvind, Team- Shlok Verma, Mayank
Guide- Runumi Devi

Date of Submission: 01-06-2023 Date of Acceptance: 10-06-2023

ABSTRACT-Cookie Clicker is a popular online game focused on students. This game requires users to perform repetitive clicks on the cookie which is an icon in the game that collects points from those clicks. Subsequently, those points are used for buying various available accessories and thus improving your progress in the game. However, these repetitive clicks create inconvenience due to the repetition cumbersome process of performing repetitive task of clicking for collecting points. This in turn affects the popularity of the game as the whole point of the game is to provide is to have points to buy accessories. Our work has implemented an AI based bot towards not only replacing the repetitive clicks but also collecting optimum relevant accessories based on the acquired points. A greedy-based approach is used for choosing optimum accessories as well as filtering out the overpriced accessories by the bot. The bot has been tested for 20 instances of game playing and is found to have achieved optimum accessories and repetition reduced by 95%.

Keywords- cookie-per second (cps), web scrapping, time-module, Web-driver, Selenium.

I. INTRODUCTION

Cookie Clicker is an incremental game created by French programmerJulien "Orteil" Thiennot in 2013. The game require user to initially clicks on a big cookie icon on the screen, earning a single cookie per click. Once earned, cookies are spent on purchasing accessories such as cursor, grandma, factory, mine, shipment, Alchemy lab, Portal, Time Machine. Each purchase further allows user to earn more cookies.Although this game allows user to earn infinite number cookies and subsequent accessories, users may aim to reach milestone numbers of cookies within a specified time.The game is played for collecting cookie by the clicking on the cookie icon. The number of cookies collected is utilized as a currency for buying the accessories like cursor, grandma, factory, mine, shipment, alchemy lab, portal, time machine.

Each accessorieshave two properties such as price and cookie generation. While buying accessories, through earned cookies the property "price" gets increased. The asse bought through the earned cookies. Each generation increases cookie-per second (CPS). However, it doesn't change the accessories cookie generation rate but the price for each accessory increases the more it's bought. Thus, after a period of time making that accessory over-priced for value provided. Thus, the goal of the game is to get the highest cps possible in for a given time.

A big number of modern RPG (role playing games) is based on the principle of autopay which allows the user to only focus on acquiring accessories of the game. This kind of game follows the revenue model that is based on player-retention with the dopamine psychology of constant reward system. Our works focuses on increasing the player retention by removing the process of manually collecting the cookies and reducing the long-playing hours. The bot simply provides user the benefit of purchasing accessories for progressing further in the game.

Our research addresses three major problems of the game such as i) the significant amount of clicking hours in the game, ii) over-pricing of the items and iii) choosing the right accessories for the current cooking credit.

The aforementioned issued are handled by achieving maximum cps. To achieve this, our research accomplishes the following task

- Automating cookie clicks.
- Filtering out the over-priced items.
- Optimum accessory selection

NOTE- The term items and accessories used in the paper refer to the same list of purchasable accessories present in the game.

II. RELATED WORK

Game automation, Bot design, and Game testing are the three domain that this research paper expands upon.

A. Game Automation[3][4][5]

Gaming automation refers to the use of software or hardware tools to automate certain tasks or actions in video games. This can include anything from automatically farming resources or leveling up characters, to performing complex actions or movements with precision and speed. Players who want to save time and effort, or who want to achieve certain goals more quickly than they would be able to through manual play often use gaming automation. The essential ideas from the associated works are as follows:

As described by Bellemare, M. G., et al the Developing AI agents with general competency across a variety of Atari 2600 games is a task that the Arcade Learning Environment (ALE) evaluation platform puts forward. It allows a range of diverse problem contexts, and the scientific community has been focusing on growing attention to it. That has led to several high-profile success stories including the highly publicized Deep Q-Networks (DQN).

The overall utilization of the ALE by the research community is looked by Bellemare, M.G, et al. They illustrate the evolution of the ALE's evaluation procedures and identify several major issues that have been addressed. Through the help of this discussions, that we can offer some methodological best practices and in their natural state benchmark outcomes. In order to continue the field's advancement, which provide an updated version of the ALE that supports numerous modes of play and provides a type of stochasticity that we refer to as sticky actions. This broad analysis brings to a conclusion by an overview of the challenges observed when the ALE initially was suggested, an overview of the state of the art in a number of fields, and an overview of unresolved problems.

Evolutionary Computation: Hingston et al. talk about the basis for developing agents being evolutionary algorithms, such as genetic algorithms or genetic programming. In order to find the best agent behaviors, these algorithms use Darwinian evolution-inspired concepts including selection, crossover, and mutation.

The features of game automation help to solve the adversities as in form of gaming while helping in the continuous trigger's that is same through the gaming process. That is Auto-aim: This feature supports a player in automatically targeting at targets, making it easier to shoot and remove competitors.Auto-fire: With this feature, a player can engage in combat frequently without having continually tapping the fire button. Auto-loot: This option helps the player save time and effort by grabbing loot from containers and enemies automatically.Auto-run: With this function, a player can run without needing to keep the run button depressed.Auto-save: This function prevents the player from having to worry about manually saving the game by automatically saving the game progress at specific checkpoints.Auto-quest: This function helps the user achieve quests and objectives by guiding them.

B. Bot Designing [6]

Bot designing is the process of creating software programs that can perform automated tasks or actions without human intervention. Bots can be designed for various purposes, such as automating customer service, data collection, social media management, and gaming automation.

The process of designing a bot involves several steps, including:Identifying the task or action to be automated, Determining the platform or environment in which the bot will operate, Selecting a programming language and development framework, Designing the bot's user interface and interaction flow, Implementing the bot's functionality and logic, Testing and debugging the bot, Deploying the bot to its intended environment, Bot designing requires skills in programming, software development, and user experience design. It is important to consider ethical and legal implications when designing bots, as some automated actions may be considered unethical or illegal.

The main key components of a test like this are divided into the following categories:

Bot Architecture: Describe the bot's architectural layout, including the main framework, individual modules, and interactions between the various parts. Talk about the architectural frameworks or patterns that were selected for the bot's development.

Sensing and Perception: Investigate the techniques and tools used in sensing and perception, such as sensor integration, natural language processing, and computer vision. Talk about the methods utilized to get pertinent data from the environment or user inputs.

Decision-Making and Control: Talk about the methods and algorithms the bot uses to make decisions and control itself. Rule-based systems, machine learning techniques, or hybrid strategies could be used in this. Describe how the bot weighs alternatives, chooses actions, and modifies its behavior in response to input or shifting circumstances.

Learning and Adaptation: Analyze the bot's capacity for learning and adapting. Talk about using machine learning, reinforcement learning, or other methods to help the bot get better with experience and feedback over time.

Integration and Deployment: Discuss how the bot will be integrated into the target system or environment. Deal with any technical issues or difficulties relating to infrastructure for deployment, scalability, security, or compatibility.

Performance and Evaluation: Analyze the bot's performance in terms of usability, effectiveness, dependability, and user happiness. Talk about the measures used to evaluate the bot's performance and any tests or empirical findings that support it.

An in-depth review of the technological factors involved in creating intelligent bots is provided by a technical analysis of bot designing. It takes into account a number of things, including architecture, sensing, decision-making, learning, communication, integration, performance, and ethical concerns. Depending on the particular environment and goals of the bot design, the precise material and level of analysis may change.

C. Game Testing[7][8]

Game testing is a process of evaluating the quality and functionality of a video game before its release to the public. The goal of game testing is to identify and report any bugs, glitches, or other issues that could negatively affect the player's experience. A team of testers who play the game in various ways and scenarios to uncover any potential issues typically conducts game testing.

Game testing typically involves several phases, including:Alpha testing: This is the initial phase of testing where the game is tested by a small group of testers to identify major issues and bugs.Beta testing: This is a more extensive phase of testing where the game is tested by a larger group of testers to identify any remaining issues and gather feedback on gameplay and user experience.Release candidate testing: This is the final phase of testing where the game is tested to ensure that it is ready for release to the public.

Test automation tools, manual testing, and exploratory testing are just a few of the methods and tools that game testers employ to find problems. Additionally, they create bug reports to record their discoveries, and they collaborate with the game's production team to fix any problems before the game is launched.

By Woof, W. and Chen, K. finding that the establishment of general video-game AI has proven to be an effective utilisation of deep reinforcement learning (DRL). Nevertheless, almost all of existing DRL audio-visual-game agents learn from the start to the end from the game's video output, which is redundant for many applications and introduces a number of additional problems. Furthermore, working directly with pixel-based basic video data is significantly different from what a real player would do. In this study, we provide an original approach that allows DRL agents to directly learn from object data.

This is achieved by using an object embedding network (OEN), that simultaneously meets the DRL and compresses an array of object feature vectors of different lengths into a single fixed-length unified feature vector representing the current game state. By evaluating the OEN-based DRL agent against a number of innovative techniques on a game selected in the GVG-AI Competition. Based on our experimental findings, the performance of our object-based DRL agent is on par with the methodologies we utilized in our comparing research.

III. PROPOSED MODEL

We have dealt with the above problem with a three-phase model, which divides and solves the problem in three phases.

1. Automating the Cookie Clicks-

Cookie clicks are automated by simply using selenium in-built functions to simply click the cookie continuously by putting it in a while loop which runs continuously, then, we use the time_constraint input taken in by the user to use as a time limit to end the continuously running loop.

2. Filtering out the over-priced items-

We have defined a list of upper limit cost for each accessory in the game and if, the price of the accessory exceeds a certain limit, it gets blacklisted from buying.

3. Optimum Accessory Selection-

After the filtration process, the list of accessories that pass the filtration process are then sent to the accessory selection process, where we aim to but the most expensive accessory in the list which we can afford to buy according to current cookie count.

Thus, this process happens in two stages-

a. Getting the list optimum items.

This is simply the list of items we get after the filtration process.

b. Selecting the affordable item from the list of optimum items.

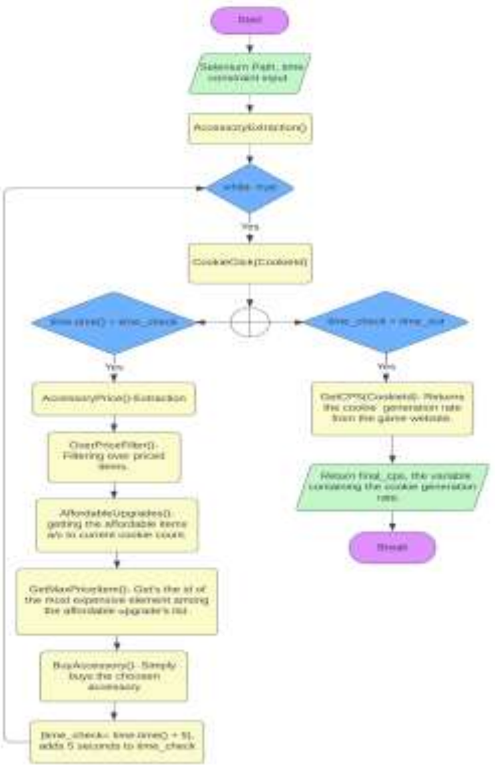
We use the list of items we got from the filtration process and the selenium in-built function to derive the current cookie count to create a new list of items which are currently affordable.Then, we select the most expensive element from that newly created affordable items list to get the highest cookie yield possible.

The processes 1 and 2 are repeated with a 5-second window gap to deal with bottleneck of the website server as well as giving the bot time to collect adequate number of cookies. If there aren't any accessories currently affordable then, the affordable accessories list remains empty and no item is purchased.

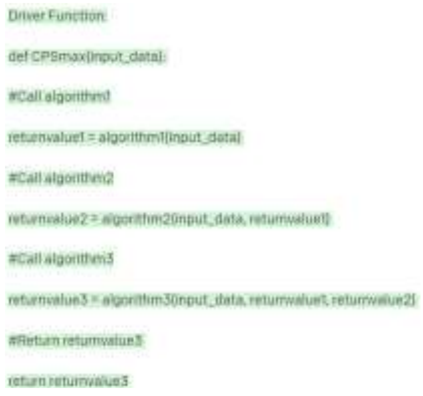
This whole above 1, 2 and 3 processes happen within the user given time_constraint input and the efficiency of the bot or that session is determined by the rate of cookie generation achieved at the end of the session.

The session ends when the time limit given by the user is over and time module has been used to keep track of the time by our bot.

Thus, our bot aims to maximise the CPS or Cookie generated per second to deal with our three problem statements.


Start
Selenium Path, time constraint input
AccessoryExtraction()
while true
Yes
CookieClick(CookieId)
time.time() > time_check
time_check > time_out
Yes
Yes
AccessoryPrice()-Extraction
OverPriceFilter()- Filtering over priced items
AffordableUpgrades()- getting the affordable items w.r.t current cookie count
GetMaxPriceItem()- Get's the id of the most expensive element among the affordable upgrade's list
BuyAccessory()- Simply buys the chosen accessory
[time_check= time.time() + 5], adds 5 seconds to time_check
GetCPS(CookieId)- Returns the cookie generation rate from the game website
Return final_cps, the variable containing the cookie generation rate
Break
Figure 1: Flow chart of algorithm CPS_Max

Suppose1, 2 and 3as algorithm1, algorithm2 and algorithm3. Returnvalue1 is used in the algorithm2 and algorithm3. Returnvalue2 is used in algorithm3. Returnvalue3 is used in our CPSmax algorithm thus, there are three major functions and one driver function that drives all three functions.

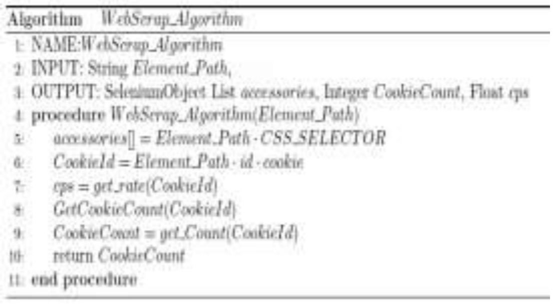


```
Driver Function

def CPSmax(input_data):

#Call algorithm1

returnvalue1 = algorithm1(input_data)

#Call algorithm2

returnvalue2 = algorithm2(input_data, returnvalue1)

#Call algorithm3

returnvalue3 = algorithm3(input_data, returnvalue1, returnvalue2)

#Return returnvalue3

return returnvalue3
```

At the end of the process, the effectiveness of the bot or the session can be evaluated by comparing the number of cookies generated within the given time constraint with the rate at which similar tasks can be completed manually.

1.1 Algorithm Approach

The algorithm is divided into four major components to achieve the three primary tasks for cps maximization. Web scrapping algorithm, Over price filter algorithm, Optimum accessory selection algorithm and CPS maximization algorithm.



```
Algorithm  WebScrap_Algorithm
1: NAME:WebScrap_Algorithm
2: INPUT: String Element_Path,
3: OUTPUT: SeleniumObject List accessories, Integer CookieCount, Float cps
4: procedure WebScrap_Algorithm(Element_Path)
5:     accessories[] = Element_Path · CSS_SELECTOR
6:     CookieId = Element_Path · id · cookie
7:     cps = get_rate(CookieId)
8:     GetCookieCount(CookieId)
9:     CookieCount = get_Count(CookieId)
10:    return CookieCount
11: end procedure
```

Input-[String Element_Path]is the system path to the selenium tool installed in the computer to activate and link the website to the driver.

Accessories []- We extract the list of purchasable accessories in the form of selenium object using the CSS_SELECTOR methodology and selenium inbuilt function and store them in the accessories[] list.

CookieId-We extract the access id of cookie icon in the game using the selenium tool and store it in the CookieId variable to be used in the CPSmax algorithm.

cps-We store the extracted cookie rate from the website in the cps variable.

GetCookieCount(CookieId)-This function returns the current cookie count or cookie earned in the game, that is later used to filter out affordable accessories to buy. It uses the extracted CookieId to access the website elements.

```
Algorithm    OverPriceFilter
NAME:OverPriceFilter
INPUT: List $WebScraper \cdot accessories[]$
OUTPUT: List $O\_Accessories[]$
procedure $OverPriceFilter$(List $acc[] = WebScraper \cdot accessories[]$)
    Integer $N = 0$
    List $Accessories\_Price[], O\_Accessories[]$
    $oLimit = [5*10^2, 2*10^3, 7*10^3, 5*10^4, 10^6, 10^8, 248*10^6, 1]$
    Integer $Length = get\_length(acc[])$
    $Accessories\_Price[] = get\_price(acc[])$
    while $N < Length$ do
        if ($Accessories\_Price[N] \leq oLimit[N]$) then
            $O\_Accessories \cdot append(Accessories\_Price[N])$
            $N = N + 1$
        end if
    end while
    return $O\_Accessories[]$
end procedure
```

Input-We use the selenium object list accessories[] derived by WebScraper algorithm for filtration process.
acc[]-Used to store the accessories[] list extracted from the WebScraper function which is called as pre-defined function parameter in the OverPriceFilter function.

Accessories_Price[]-List that simply stores the current extracted prices of the accessories by extracting the prices of the accessories from the list of selenium object- accessories[].

oLimit[]- We have calculated the upper limit for the prices of each of the accessories exceeding which the element would be blacklisted and won't be counted in the above accessory selection.The limit has been set in a manner that if, the price of an accessory exceeds the price of the next-higher-expensive accessory keeping their base price as reference point, then, the lower accessory would be blacklisted and the next higher accessory would be selected.
In case of the last accessory the "time machine" we have set no-limit and have simply assigned '1' as it's the most valuable accessory and there's no other higher accessory present.

Accessories	**Upper Price Limit**
Cursor	5*10^2
Grandma	2*10^3
Factory	7*10^3
Mine	5*10^4
Shipment	10^6
Alchemy Lab	10^8
Portal	248*10^6
Time Machine	1

O_Accessories[]- Simply stores the list of accessories that pass the over-price filtration process.

```
Algorithm    OptimumAccessorySelection
NAME:OptimumAccessorySelection
INPUT: List $temp = OverPriceFilter()$
OUTPUT: List $affordable\_acc[]$
procedure $OptimumAccessorySelection$(List $temp = OverPriceFilter()$)
    List $affordable\_acc[] = NULL$
    Integer $x = WebScraper \cdot GetCookieCount()$
    $N = 0$
    $Length = get\_length(temp[])$
    while $N < Length$ do
        if ($x \leq temp[N]$) then
            $affordable\_acc \cdot append(temp[N])$
            $N = N + 1$
        end if
    end while
    return $affordable\_acc[]$
end procedure
```

Input- O_Accessories[] returned by OverPriceFilter function called as a pre-defined function parameter and is stored in temp[] list.
List affordable_acc[]-List that stores the prices of the items that are currently affordable according to the current cookie count, this list remains empty if, no items are purchasable.
Integer x-Stores the current cookie count returned by WebScraper.GetCookieCount function. If, the current cookie count is less than the price of a accessory then the accessory won't be added in the affordable_acc[].

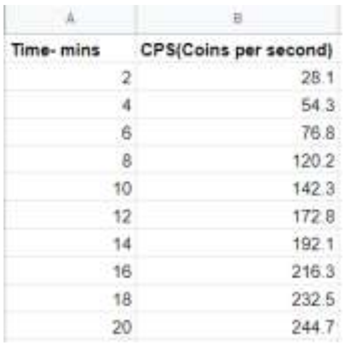
```
Algorithm    CPSmax
NAME:CPSmax
INPUT: String $Path$ , Integer $time\_constraint$
OUTPUT: Float $cps$
procedure $CPSmax(Path, time\_constraint)$
    List $cps$
    String $ToBuy$
    List $affordable\_acc[]$
    Float $time\_out = (time\_constraint * 60) + time \cdot time()$
    Float $time\_check = time \cdot time() + 5$
    while $true$ do
        CookieClick(WebScraper.CookieId)
        if ($time \cdot time() > time\_check$) then
            $affordable\_acc = OptimumAccessorySelection()$
            $GetMaxPriceItem(affordable\_acc[])$
            $Purchase(ToBuy)$
            $time\_check = time \cdot time() + 5$
        end if
        if ($time\_check > time\_out$) then
            $cps = WebScraper \cdot cps$
        end if
    end while
    return $cps$
end procedure
```

Input-String Path is the selenium driver path passed to the algorithm to connect with website and Integer time_constraint is the user allotted runtime to get the highest CPS possible in that runtime.

List cps-List that stores the cps returned from the website at the end of the algorithm, that determines the efficiency of the algorithm.
String ToBuy-Stores the id of the accessory that is selectedin the GetMaxPriceItem(affordable_acc[]) function call.
time_out-The time_constraint input is converted into minutes to determine the time limit of the program run. The program is terminated returning the currently achieved cps when the time is up.
time_check-time_check ->time.time(), means it's assigned to the time module that represents the current time of the system to keep track of the flow of time in the program.
CookieClick(WebScraper.CookieId**)-** Simply performs the cookie clicks.
affordable_acc[]-Stores the list of affordable accessories returned by the OptimumAccessorySelectionfunction.
GetMaxPriceItem()-Takes the affordable_acc[] list as input and simply returns the most expensive item in the list which is stored in string variable ToBuy.
[time_check = time.time() + 5]-Increments time_check by 5 more seconds making it run faster than the currrent time, thus, [time.time() >time_check] condition becomes true only after 5-second intervals making the accessory purchase process occur at 5-second intervals. Giving the bot cookie generation time and reduces bottleneck.

IV. EXPERIMENTAL ANALYSIS

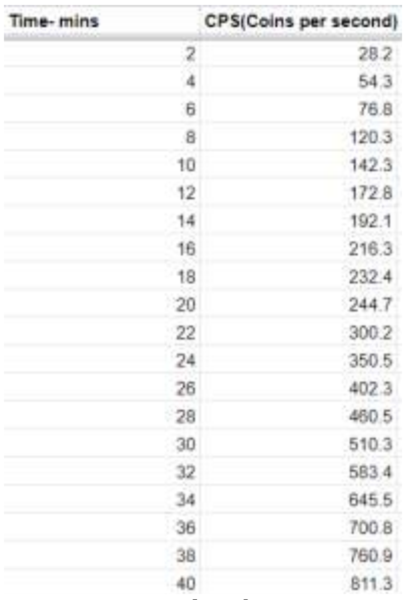
The program has been tested for 23 instances with varying input of time constraints. The output given by the program for varying input constraints are as follows:



A	B
Time- mins	CPS(Coins per second)
2	28.1
4	54.3
6	76.8
8	120.2
10	142.3
12	172.8
14	192.1
16	216.3
18	232.5
20	244.7

Figure 2: Testing results obtained by various input constraints for 20 mins

The cps achieved in the above time constraints aren't fixed but have a variation of (-5 to +5) seconds depending on the responsiveness of the website as well as the path our bot takes in each run.When the time_constraint's range is increased to 40mins, it gives a CPS with variation from (-10 to +10) and the variation increases with the increase in time_constraint.



Time- mins	CPS(Coins per second)
2	28.2
4	54.3
6	76.8
8	120.3
10	142.3
12	172.8
14	192.1
16	216.3
18	232.4
20	244.7
22	300.2
24	350.5
26	402.3
28	460.5
30	510.3
32	583.4
34	645.5
36	700.8
38	760.9
40	811.3

Figure 3: Testing results when time constraint increased till 40 mins

Factors affecting the CPS per iteration for a given time constraint-

- Variation in responsiveness of the website per extraction or action command.
- The filtered-out item
- cookie_count variation with max(affordable_items) per iteration.
- The compounding effect of all the above factors.

V. GRAPHICAL & MATHEMATICAL ANALYSIS

1.2 Graphical Analysis

Our algorithm with the input(time_constraint) gives a CPS in the pattern of n^2 where n is the time_constraint in minutes. For ($1<n<20$) the observed pattern is n^2. For the time constraint of ($20<n<40$), where n=minutes, the observed pattern is n^3.

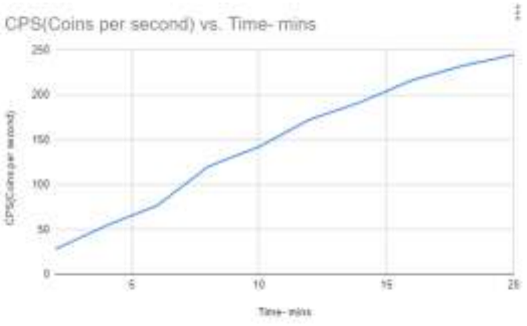


Figure 4: CPS vs Time(in mins)-20 mins graph

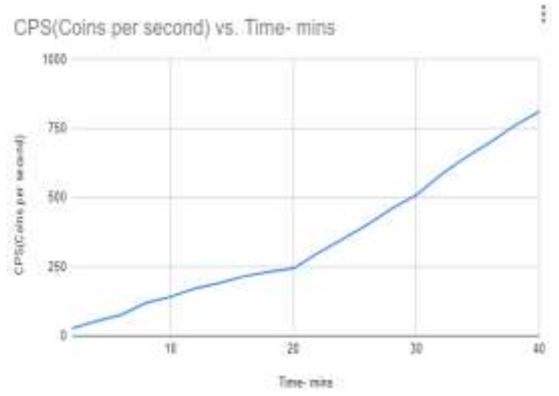
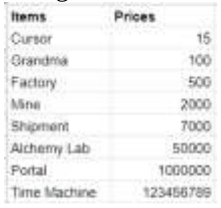


Figure 5: CPS vs Time(in mins) - 40 min graph

Getting the graphical pattern of item prices and the increase in the pricing of the items.



Items	Prices
Cursor	15
Grandma	100
Factory	500
Mine	2000
Shipment	7000
Alchemy Lab	50000
Portal	1000000
Time Machine	123456789

Figure 6: Exponential increase in the pricing of items

As for the increasing deviation with increasing time constraints, this may be due to a range of factors, including resource exhaustion, server overload, or other external factors beyond your bot's control. To mitigate this issue, you may need to implement a more sophisticated performance monitoring system to identify and troubleshoot bottlenecks or other issues that may arise during longer runs.

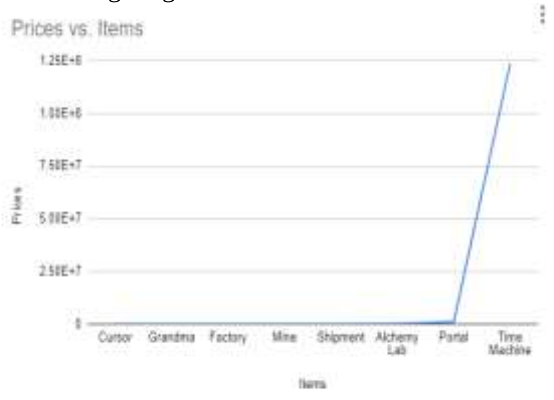


Figure 7: Pricing vs Items graph exponentially increasing

The items vs prices graph's pricing increases exponentially after reaching the portal prices and the CPS vs time graph should always be behind the price graph to be able to purchase the items after filtration. Meaning the (CPS vs time) graph for time n would always follow behind (Price vs Items) graph.Assuming you took an action to filter certain items according to the price, then the time taken to purchase the items would be shorter in comparison to purchase the same items without any filtration. The CPS vs time graph should also have an upward slope that increases with time; however, it should always be lower than the Price vs Items graph. The end result should look something like this: Price Vs Items \/ CPS Vs Time

1.3 Mathematical Analysis

Means the equation of (CPS vs time) increases for time duration (0,20,40,60…. n-20, n) creating an arithmetic series of a=0, d=20, a_n=n. The respective equations of (items vs price)graph for n time interval of 20 minutes increment gives a geometric series for price increase of items bough in time n is $n^1, n^2, n^3, n^4, \dots, n^n$.

$$n(1+n+n^2+\cdots n^{n-1}) = n\frac{n^n-1}{n-1}$$

$$S = 1+n+n^2+\cdots n^{n-1}$$

$$nS = n+n^2+n^3+\cdots n^n$$

$$S(1-n) = 1-n^n$$

$$S = \frac{1-n^n}{1-n}$$

Thus, the equation of total price for time_duration "n" is $S = \frac{1-n^n}{1-n}$ and the equation for instance n is n^n. Thus, as the function of (CPS vs time) is slower than the increase price rate by game rules [8] the geometric series for time n(mins) is $2^3, 2^6, 2^9, \dots, 2^{3n}$. In case of CPS the function remains on average constant throughout the period of time(n-minutes).Hence, for time(n minutes) cps attained by the algorithm at instance n is 2^{3n}.

CPS= 2^{3n} for n>=1& (n=time constraint)

VI. CONCLUSION

Greedy approach of per iteration comparison is the most suitable approach for the algorithm of our cookie clicker bot because we don't have the foresight to account for the website responsiveness& variation of filtered itemsas well as the max(affordable_upgrades) per iteration put together, due to lack of this foresight we can't use dynamic approach for this problem. The greedy approach used deals with the immediate problem of over pricing as well as selection of maximum price to attain the maximum cps. The game follows the increment of **n^n for time n minutes** andthe bot algorithm for CPS follows an instance increment of $\mathbf{2^{3n}}$ **for time n minutes**. Thus, we have automated the hectic process of coin collection as well as addressing the problem of overpricing of the items faced in cookie clicker game. Automated gameplay recently has become a big part of climber as well as role playing games and our bot works on a similar methodology of increasing the player retention rate by giving them the satisfaction of having their empire increase. The algorithm also frees the user from the burden of time required to purchase certain accessories as program run on user defined time_constrints making it easier for the user to have a prediction of wait time required for all the accessories.

REFERENCES

[1]. S. Fizek, "Automated State of Play-Rethinking Anthropocentric Rules of the Game," Digital Culture & Society, 2018.

[2]. M. K. R. G. K. F. S. Chia-Hui Chang, "Web Information Extraction System," in IEEE Computer Society, 2006.

[3]. Chang, C. H., Kayed, M., Girgis, M. R., &Shaalan, K. F. (2006). A survey of web information extraction systems. IEEE transactions on knowledge and data engineering, 18(10), 1411-1428.

[4]. Bellemare, M.G., Naddaf, Y., Veness, J. and Bowling, M., 2013. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47, pp.253-279

[5]. Hingston, P., & Zhang, M. (2007). Evolving agents for computer games. IEEE Transactions on Evolutionary Computation, 11(5), 556-566.

[6]. Wallace, J.R., Pape, J., Chang, Y.L.B., McClelland, P.J., Graham, T.N., Scott, S.D. and Hancock, M., 2012, February. Exploring automation in digital tabletop board game. In Proceedings of the ACM 2012 conference on computer supported cooperative work companion (pp. 231-234).

[7]. Albaghajati, A.M. and Ahmed, M.A.K., 2020. Video game automated testing approaches: An assessment framework. IEEE transactions on games.

[8]. Woof, W. and Chen, K., 2018, August. Learning to play general video-games via an object embedding network. In 2018 IEEE Conference on Computational Intelligence and Games (CIG) (pp. 1-8). IEEE.

[9]. Fernández-Ares, A., García-Sánchez, P., Mora, A.M., Castillo, P.A., Guervós, J.J.M., Camacho, D., Gómez-Martín, M.A. and González-Calero, P.A., 2014, June. Designing competitive bots for a real time strategy game using genetic programming. In CoSECivi (pp. 159-172).

[10]. Vancouver