THEORY REPORT PART B

BY SAMUEL MCKID

SYSTEM SPECIFICATIONS 2
- **SOFTWARE** 2
- **HARDWARE** 2

TEST DATA 3

PEER REPORT (DESK CHECK) 4
- **ALGORITHM I:** 4

CODING JUSTIFICATION (FLEX NATION/APOLOGISE FOR SINS) 5

REFLECTION 6

GitHub Link:
https://github.com/MCSQU1D/Test_-0054

SYSTEM SPECIFICATIONS

SOFTWARE

Operating System	MacOS Mojave (Version 10.14.6)
Language	Python (Version 3.7.6)
Library(ies)	PyGame Library (Version 2.0.14)

Currently, the software specifications are; Python (Version 3.7.6), on MacOS Mojave (Version 10.14.6). This reflects the development environment of the solution, and is therefore what is guaranteed to run properly with minimised chance of failure. Exporting the solution to work on other platforms will increase development time and resources, and thus will only be considered once a viable solution has been developed in the development environment. That being said, the operating system may not have an impact on the operation of the program, provided the language and all libraries are fully supported. The PyGame library (Version 2.0.14) is used for the graphical user interface, as python alone does not support a graphical user interface. A .txt file editor will be needed if custom molecules and reactions are desired

HARDWARE

Component	Minimum	Recommended
CPU	2GHz intel i3 or Ryzen 3 equivalent	4GHz i7 or Ryzen 7 equivalent
RAM	4GB DDR3	8GB DDR4
GPU	Intel HD 6000 or AMD RX Vega 6	nVidia GTX 1030 or AMD RX 550
Storage	4MB 5400RPM HDD	10MB SSD

Due to the un-optimised nature of the program, running it requires a lot more system resources than a polished version would, especially resource heavy on the CPU. While further polishing and redesigning the code would reduce the demand for resources and relocate to other components (i.e. RAM), this would exceed the project timeframe. An internet connection will be required for initial install, but not for normal use.

TEST DATA

From function “ButtonHover(Mouse_Position)”

```
if x > xlimithigh and x < xlimitlow and y > ylimithigh and y < ylimitlow:
```

x	y	xlimithigh	xlimitlow	ylimithigh	ylimitlow	Statement value
50	50	90	100	190	200	FALSE
150	250	90	100	190	200	FALSE
100	200	90	100	190	200	FALSE
95	195	90	100	190	200	TRUE
95	205	90	100	190	200	TRUE

From function “CreateButton(x1, x2, y1, y2, name)”, function have been replaced with printed flags

```
if name not in ["InformationPanel", "WorkSpace", "InformationMenu", "Temp_Up",
"Temp_Down", "Clear_Atoms", "Physics_status", "Info_status"] and name !=
Selected_Molecule:
    print(“AtomName”)
if name == "Temp_Up":
    print(“TempUp”)
if name == "Temp_Down":
    print(“TempDown”)
if name == "Clear_Atoms":
    print(“ClearAtoms”)
if name == "Physics_status":
    if Physics_Running == True:
        print(“PhysicsTrue”)
    if Physics_Running == False:
        print(“PhysicsFalse”)
if name == "Info_status":
    if InformationMenuToggleBool == False:
        print(“MenuFalse”)
    if InformationMenuToggleBool == True:
        print(“MenuTrue”)
if name == selected_molecule:
        print(“IsSelected”)
```

name	print	Physics_Running	InformationMenuToggleBool	selected_molecule
He	AtomName	True	True	H
H	AtomName IsSelected	True	True	H
Random		True	True	H
Physics_status	PhysicsTrue	True	True	H

From Function “Physics()”

```
if atoms[i].temperature > Atom_Dict[atoms[i].type]["Melting_temp"]:
            atoms[i].state = "liquid"
        if atoms[i].temperature <= Atom_Dict[atoms[i].type]["Boiling_temp"]:
            atoms[i].state = "liquid"
        if atoms[i].temperature > Atom_Dict[atoms[i].type]["Boiling_temp"]:
            atoms[i].state = "gas"
        if atoms[i].temperature <= Atom_Dict[atoms[i].type]["Melting_temp"]:
            atoms[i].state = "solid"
```

atoms[i].temperature	Atom_Dict[atoms[i].type] ["Melting_temp"]	Atom_Dict[atoms[i].type] ["Boiling_temp"]	atoms[i].state
-50	0	100	solid
0	0	100	solid
50	0	100	liquid
100	0	100	liquid
150	0	100	gas

PEER REPORT

ALGORITHM I:

Standard binary search

```
Low = 1
High = len(SortedList)
Found = False
ItemToFind = 30
while High >= Low and Found == False:
    Middle = int((Low + High)/2)
    if ItemToFind < SortedList[Middle]:
        High = Middle - 1
    elif ItemToFind == SortedList[Middle]:
        Found = True
    else:
        Low = Middle + 1
if Found == True:
    print("Found")
else:
    print("Not found")
```

ALGORITHM I DESK CHECK

Low	High	Found	ItemToFind	Middle	SortedList	Print
1	5	False	30	3	[5,12,30,55,70]	
	2					
1	2			1		
2						
2	2	True		2		Found

Comment:
A simple binary search algorithm. The variable names are easily understood, and the variables are easily modified. Overall good program design, however a round function may need to replace the
“ `int((Low + High)/2)` ” on line 6, although the program still works as needed without this change.

ALGORITHM II :

From Alex Richardson.

```
def create_sma(self, period, start_date):
        sma = 0

        num = start_date
        for i in range(period):
            sma += self.historical_close[-num]
            num +=1

        sma = sma/period

        return(sma)
```

ALGORITHM II DESK CHECK

period	start_date	sma	num	self.historical_close[-num] (from sample data)
5	12	0	12	700
		700	13	400
		1100	14	600
		1700	15	500
		2200	16	450
		2650	17	
		530	17	

Comment:
The algorithm is very simple, providing ‘sma’ as an average of the historical closing price of an object over a set period and date. However, comments should be added to explain the meaning of the variables as a non-financial maintenance programmer may not understand what ‘sma’ or historical close means.

CODING JUSTIFICATION

Due to my experience in both Python and the graphic interface PyGame, it was an obvious choice to reduce time spent on learning, and maximising time to produce a high quality finished simulator. Additionally, the object oriented paradigm was very important, as each molecule would need to be represented as an object with class features. Although there are sections of code in which the logic paradigm would be beneficial, those are relatively small, and can be effectively written in the object oriented paradigm. In hindsight, the language and paradigm choice where both correct, and helped improve the standard of the finished product

The style of FileA is honestly average, mostly due to the limited time allocated to the title screen. The dual file system was important to reduce the size and computing of FileB, and make testing the simulator easier. The main focus was to quickly create a simple, easy to understand title screen that can display the legal disclaimer. Although a few modules are shared between FileA and FileB, these mostly originated from FileB. While the style could be better, overall FileA serves its purpose well as a title screen.

In contrast to FileA, FileB is much more modular, stemming from the development process. Firstly, the basescreen was quickly developed to spend as long as possible on the physics and chemistry aspects of the simulator. Following this, the physics module was developed in a separate file, although in a very modular state. This was to make the integration of these files as easy and quick as possible, with minimised chance of errors. Although chemistry was originally planned to be a third, separate module, the physics module had the right function [Stacking(atom_A, atom_B), found atoms that hit each other] used for the chemistry module. This is why only physics is called in the main loop, as the chemistry module is a branch of the physics module. Although there are limitations to this design, overall it has been successful, and allowed the time frame to be achieved. Another reason for the highly modular design was the possibility of threading to reduce system resource requirements, but this was ultimately scrapped due to time constraints, as there were issues with variables (deleted certain atoms, while another module was printing).

Features:

Classes/Particle system

The use of a main class that defines the traits of the atoms was extremely beneficial to the development of the program. The storage of all molecules as objects in a list made management and modification of a molecule's values incredibly easy and simple. Additionally the use of a dictionary to store data on various molecules made retrieving data simple. These features allow for retrieving specific data about molecules easily done in one line:

```
atoms[i].temperature > Atom_Dict[atoms[i].type]["Melting_temp"]
```

.txt files

The use of files to store information regarding reactions and molecules allows for users to create custom molecules and reactions, as well as modify existing molecules and reactions. This also allowed for easy testing and debugging of physics and chemistry modules.

Physics and Chemistry

The Physics and Chemistry module is the main feature of the simulator, but despite the heavy time investment into both, there are oddities existing within both. There was a major issue with overlapping particles (sharing the same coordinates), although this has mostly been minimised. A system which prevents the sharing of coordinates would be best, the idea of eventual consistency was chosen due to its easier implementation (i.e. eventually no particles will share the same coordinates). This also took time to develop and create, so despite the lack of large code, there were several iterations and versions.

REFLECTION

The planning and preparation for the development of this project was successful, and progress began without any issues. The Gantt chart made during part A guided the time allocation of development, ensuring that progress remained on track. One of the main focuses was to reduce time on GUI development, as over allocation of time to this area has been a problem in previous projects. This was successful, and minimum time was spent on creating a high quality GUI, the majority of visual elements were finished within the first week. Another strong point is the file system, as advanced users can modify aspects of the game to their liking, something that is highly valuable to many users. However, this does come with risks of failure if edits are incorrect, something that may be improved. Other areas for improvement are the physics system, as there are several issues with the current system. One of the biggest problems encountered during development is making sure that the particles do not overlap (no 2 particles share the same x and y position). While minimised, this issue still exists in the current system. Other issues are the questionable liquid state, and introducing a non-particles solid state (permanent position on workspace). The chemistry module also has issues, mainly with being able to create multiple particles, or use more than two particles in a reaction. The chemistry module also prevented correct integration of threading, something that would have massively reduced the load on system resources, specifically CPU time. Despite the shortcomings that were to be expected of a project developed with the limited resources and time constraints, the project as a whole is a solution of high quality that satisfies the criteria of its development.

From the developer:
Personally I am proud of how this project has developed, expanding my skills in many aspects of Python programming and project management. The Gantt chart and system diagrams created in Part A simplified the development process, as these could be quickly developed due to the created designs, providing a base for the program to be developed off. Although not completely how imagined, the program is, in my arrogant opinion, a high quality piece of software.

Sam ‘MCSQU1D’ McKid.