Oliver Askew, Jack Marschall, Erfan Safiri a1852800, a1851490, a1852282

The Monarch

Use case description of the game and user interface.
The monarch is a role-playing game where the user is the king who is presented with problems by his advisors. The user can type “y” or “n” on the complier to either accept or reject the advisor’s suggestion. The Users answer will affect different pillars of society such as army, church, wealth, and people. These pillars must be kept at balance as when the values of a pillar become zero or is maximised, the user loses. The user is then required to enter a name for the new king and continue the game as a new king.
After each decision, the value for each pillar is updated and presented in the text format to the user. Each advisor is depicted with ASCII art, the advisors include the General, Queen, Treasurer, Excursionist, Spirit, and the Pope. Below the ASCII of the character, a text containing the problem is shown on the compiler.

Description of classes

Person - should be used as an abstract class where advisor and other classes are inherited from Person. A person in the game can only show_Face().

Advisor – Could be banker, general, king’s guard and queen. They can speak to the king.

King – which is the user, can decide, and his decisions will have impact on pillars of power. Once king is dead, a new king object is created by the user. User can set the king’s name. The name must be different than the previous king’s name.

Reduce_value and increase value () are used, to increase or decrease a certain variable of a power such as church or Army.
The power of king matters because the impact of a decision is calculated from
Power * Reduce_value, Power* increase _value. This is to make the power of king and advisors dynamic and dependant on the game progress. -

Pillars of Power - In each power, the overall_ value is calculated from other variables. The values are identified by the below symbols.

Church – satisfaction value is set at max 100, it is calculated from number of churches and wealth of pope. Each of those variables are set at 50.

Wealth – Similar to church
Army – Similar to church
People – Similar to church

Card(scenario) - In each scenario a text and options are presented to the user. There will be an if statement, if a certain answer is chosen certain values are deducted or increased.

Consequence Class - This is intended to be activated when one of the overall values are reached to zero and the king is killed or died. This class will show text of how king died followed by a visual.

Time.cpp – This is like the main.cpp. It is the storyline, where different classes and functions are called to progress the game.

To Do and Timeline

- Story line, content of up to 10 mins of play time by 10 oct
- Visualisations, how will the characters look, how will the user choose options by 13 oct
- Make the class designs and inheritances more refined. To meet the requirements, of inherence from A -> B ->C. by 1 oct

Writing Code for Classes (by the 5th):

- Person
- Advisor
- King
- Pillars of power
- Church
- Wealth
- Army
- People
- Card(scenario)
- Consequences

Unit testing each class by 15 oct

Plan for unit testing

Test each class. If error is shown, tester can check alongside the person who wrote the class to have an outside perspective.
Once all the individual classes are tested, the program as a whole will need to be tested and with how the classes interact by each independent person using a different system, through a meeting we will discuss the errors and areas of improvement. Unsolved errors that arise from certain sections are given back to the original programmer to debug that section.

- As there are limited inputs user error should be easily limitable
- Testing of each file can be conducted through altered, simple versions of the main function and checking that each class can interact as intended
- Completing playthroughs
- Achieving different deaths e.g., max or min values

	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
Week 8				Research and idea time/Timeline creation			
Uni Break 1	User description		Class creation				
Uni Break 2	Class creation				Planning for testing		
Week 9	Compilation and checking			Coding	MARKING	Coding/Testing	
Week 10	Coding/Testing						
Week 11	Debugging and Checking				MARKING