Plan of Attack

Break Down:	Member:	Finish Date
1. UML & Plan of Attack	Liam & Fanny	11/24
2. Controller (responsible for read in command from a saved file)	Fanny	11/25
3. Board Class (exclude trade, mortgage, bankruptcy)	Liam	11/26
4. Player Class	Liam	11/27
5. Cell Class & Dice Class	Fanny	11/26
6. Ownable & Academic Class (exclude improvement, trade)	Fanny	11/27
7. Unownable & its subclass(exclude slc and needles hall)	Liam	11/27
8. Gym and Residence Class	Liam	11/28
9. Debug program together	Liam & Fanny	11/29
10. View & TextDisplay	Fanny	11/30
11. Finish bankruptcy and mortgage	Liam	11/30
12. Finish SLC and Needles Hall	Liam	12/1
13. Finish Improvement	Fanny	12/1
14. Finish Trade and display part	Fanny & Liam	12/2
15. Debug program and fix other bugs	Fanny & Liam	12/4

Question Answers:

For Q1:
Observer is a good pattern for implementing the game board. The board is a subject and each cell on the board is an observer of the game board. Every time when a player rolled a dice, it will notify all the cells that the player is going to pass and then for ownable buildings is also a subject itself. Every time an ownable building is bought it will notify its neighbors. For example, if CIF is bought then it will notify PAC the owner of CIF. Similarly, if RCH is bought, it will notify DWE, CPH.

For Q2:
We think use template pattern is suitable for implementing SLC and Needles Hall object. They are both the same type of class and the only difference is their effect so we can use template pattern to handle it.

For Q3:
In our solution, we have a field in our player class which is the number of Tim cards they owns. And in our board class we have a method which adds each player's number of Tim cards together and make sure it is less than four. Moreover, in our cell class, we have a pointer to the board class so that every

time when we need random a card, we will call this method from board to check if we can create a new Tim card or not.

For Q4:
Decorator is not a good way to implementing improvements. We think we can just add a few fields in the Academic Building class to handle this improvement situation. For example, we will have an integer filed to count the improvement level of this building. Therefore, every time when a player buys an improvement the value of this building will increase and meanwhile, it will increase the rent that other players need to pay for landing on this square. Similarly, when they sell the improvement functions in the Building class will decrement accordingly.