2. design pattern :

- composite : מבנה כגון Pack הוא Composite כיוון כי יורש של כמה אירועים את events, שאר האירועים הם leafs (structual)

- strategy : בכל סוגי מחלקה יש job ו-Character, כי שחקן Player יכול להכיל כמה סוגי Character וכל אחד שונה, וגם יש כמה סוגי job שונה מחלקה. יש שוני של האסטרטגיה של Character כאשר הם משנים כולם את ההתנהגות מ- Job או Character.

3. בונוס מחלקת Rogue

אין צורך לשנות את ה design - מאחר ויש רק לממש מחלקה חדשה ויורשת ל-Job בשם Rouge כמו כל הילדים של Job ולכן יש כמו מחלקת warrior מימוש שיכול שונה הדרך שבו שונה getcombat power ולכן מימוש ייחודי של Rogue מוכלל בזה.

```cpp
int Rogue::getcombat(Player& player) {
    int combat = (player.getForce()) + player.getLevel();
    return combat;
}
```

ועלינו צריך לשנות את ההתנהגות ה"ל גם כן :

```cpp
void Player::applyEncounter(const Event& event){
    int playerCombatPower = getCombatPower();
    int eventCombatPower = event.getCombatPower();
    int loot = event.getLoot();
    int damage = event.getDamage();
    if(playerCombatPower > eventCombatPower){
        this->LevelUp();
        this->addCoins( getcoins: loot);
        cout<<getEncounterWonMessage( player: *this, loot)<<endl;
    }
    else if(m_job.getJob() =="Rogue" && eventcombatPower > 2*playerCombatPower )
    {
        return;
    }
    else{
        this->damage( damageHP: damage);
        cout<<getEncounterLostMessage( player: *this , damage)<<endl;
    }
}
```

4 אזרח טוב : כן , לולו אחלק עכשיו טבלה מעלה n_event כמו DivineInspiration, תארו בעזרת הקשרים השונים

הסבירו ויהי עבור הפעלת applyevent בקלטר את סו בכל זורו הי בכלל ביכא

```cpp
class DivineInspiration : public Event{
public:
    void applyCard(Player& player) const override {
        srand(time(nullptr));
        int randomIndex = rand() % availableJobs.size();
        player.setJob(availableJobs[randomIndex]);
    }
private:
    vector<Job*> availableJobs;
};
```