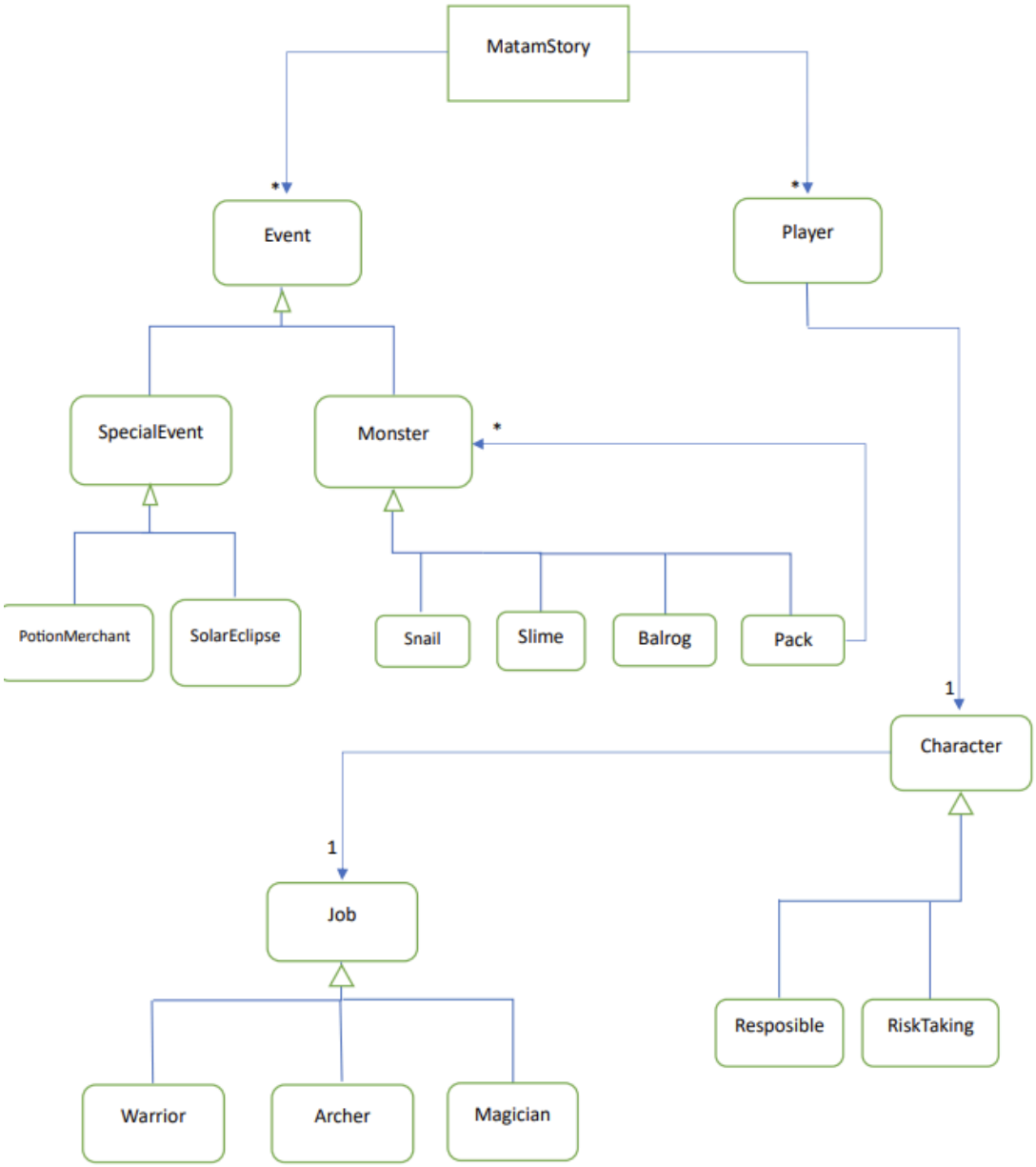
מוחמד ברכה	214164857
כרם בשארה	214697435

UML

שאלה 1:



שאלה 2:

כן, השתמשנו בשלוש תבניות עיצוב (Design Patterns):

Strategy במחלקות Player ו-Character:
השתמשנו בתבנית העיצוב Strategy, שהיא תבנית מסוג התנהגותי (Behavioural Design Pattern), מכיוון שכל שחקן (Player) פועל בהתאם לקריטריונים מסוימים, כלומר, מתנהג בהתאם לדמות (Character) שלו, וכל דמות מתנהגת לפי העבודה (Job) שלה.
המשמעות היא שכל שחקן (Player) מקושר לדמות (Character), וכל דמות מקושרת לעבודה מסוימת (Job).
תבנית ה-Strategy מאפשרת התנהגות שונה עבור שחקנים בהתאם לדמויות ולעבודות השונות שלהם.

Composite:
השתמשנו בתבנית Composite, שהיא תבנית מסוג מבני (Structural Design Pattern), מכיוון שהיא מסייעת בהרכבת אובייקטים ומחלקות.
לדוגמה, במחלקת Monster יש מחלקה יורשת בשם Pack, המחזיקה אובייקטים מטיפוס ההורה שלה (Monster).
תבנית ה-Composite מאפשרת לבצע זאת בצורה פשוטה ומובנית.

Factory:
כדי לטעון אובייקטים מתוך קבצי קלט (כגון Players ו-Events), השתמשנו בפונקציות שמבצעות התנהגות דומה לתבנית העיצוב Factory.
פונקציות כמו EventInventer, MonsterInventer, ו-PlayerInventer קוראות לקובץ המתאים, מזהות את שם הטיפוס, ויוצרות מצביע חכם לטיפוס זה.
יש לציין שלא מימשנו תבנית Factory בצורה מדויקת (לא יצרנו מחלקת Factory ייעודית), אלא השתמשנו בהתנהגות דומה יחסית לעקרונות של תבנית זו.

שאלה 3:

יש להוסיף מחלקה חדשה בשם Rogue, שיורשת מהמחלקה Job (class Rogue : public Job).
במחלקה זו יש להגדיר בנאי שמקבל רק את שם השחקן ומאתחל את המשתנים coins, level, ו-force באופן אוטומטי, בהתאם לערכים הנדרשים לגנב במשחק.

כמו כן, יש להפוך את הפונקציה Encounter שבמחלקה Job לפונקציה וירטואלית.
במחלקה Rogue, יש לכתוב פונקציה שמבצעת override ל-Encounter ולממש בה את ההתנהגות של הגנב כאשר הוא נתקל במפלצת, לפי הדרישות במשחק.

שאלה 4:

תחילה, נכריז על משתנה מסוג MAP_Unordered במחלקה Job בתור static, כך שהוא יהיה משותף לכל סוגי העבודות השונות. המפתח במבנה הנתונים הזה יהיה מסוג int, והערך יהיה functor מסוג <std::function<Job*(std::string)>, שמאפשר יצירת אובייקט חדש מטיפוס עבודה מסוים (לדוגמה: new Warrior).
נגדיר באופן ידני שמספר 1 מייצג Warrior, מספר 2 מייצג Archer, וכן הלאה.
בנוסף, תוגדר פונקציה static שתבנה את ה-Map בפעם הראשונה שבה נוצר אובייקט שיורש מהמחלקה Job. לאחר יצירת ה-Map, הגישה אליו תתאפשר לקריאה בלבד.

ניצור מחלקה בשם DevineInspiration, שיורשת מהמחלקה SpecialEvent (class DevineInspiration : public SpecialEvent).
במחלקה זו נוסיף את הפונקציות הבאות:

פונקציית override ל-getDescription – לממש את הפונקציה מאחר שהיא מוגדרת כ-pure virtual במחלקה SpecialEvent.
פונקציה SetJob – שמשנה את העבודה של השחקן על ידי מחיקת העבודה הקודמת.
פונקציית override ל-EncounterEvent(Player) – בפונקציה זו נשתמש ב-random של C++ כדי לבחור מספר רנדומלי בטווח שבין 1 למספר איברי ה-Map. לאחר מכן, ניגש לערך המתאים ב-Map באמצעות המפתח הרנדומלי, נפעיל את ה-functor, ונשתמש ב-SetJob כדי לעדכן את העבודה של השחקן לערך שהתקבל מה-functor.
כיוון שהפרמטרים Force, Level, HP, ועוד נשמרים במחלקה Job, ואם נדרש שהעבודה החדשה תשמור על אותם ערכים (כלומר, שהשחקן לא יתחיל ממצב של "שחקן חדש"), נוסיף למחלקת Job פונקציות getters ו-setters עבור הפרמטרים הרלוונטיים.
בנוסף, לפני מחיקת העבודה הקודמת ב-SetJob, נתאים את הערכים של העבודה החדשה כך שישמרו על ערכים תואמים לעבודה הקודמת.