**C++ Paradigm:**

The solution effectively follows the object-oriented programming (OOP) paradigm by utilising several key language features of C++ :

**Encapsulation:**

The Player class encapsulates the player's balance, ensuring data integrity by using access control for private balance attributes and public methods.

**Abstraction:**

The SlotMachine class acts as an abstract base class, demonstrating abstraction by defining common behaviors and leaving the implementation details to its subclasses, NormalMachine and JackPotMachine.

**Inheritance:**

The inheritance relationship between the SlotMachine base class and its subclasses promotes code reuse and hierarchy, allowing subclasses to implement their variations of abstract virtual functions.

**Polymorphism:**

The chosenMachine pointer, which is of type SlotMachine but can point to either a JackPotMachine or a NormalMachine object, showcases polymorphism by allowing different objects to respond differently to the same method calls.

**Object Composition:**

The GameController class has a composition relationship with the Player object, as it contains an instance of the Player class as a member variable, demonstrating object composition.

The approach complies with the OOP paradigm by utilising good use of these language properties. By encapsulating data, abstracting common behaviours, encouraging code reuse through inheritance, permitting polymorphic behaviour, and combining objects to create bigger systems, it clearly exhibits a grasp of OOP principles. Additionally, the solution places a strong emphasis on modularity, which promotes extensibility and maintainability by giving the slot machine game a clear, adaptable structure. Overall, the solution adheres to the defined paradigm and demonstrates a useful implementation of OOP in C++.