The Use Case

The lesson provides the implementation of a particular use case of std::variant and std::optional

Consider a function that takes the current mouse selection for a game. The function scans the selected range and computes several outputs:

- the number of animating objects
- if there are any civil units in the selection
- if there are any combat units in the selection

The existing code looks like this:

```
class ObjSelection
{
public:
 bool IsValid() const { return true; }
 // more code...
};
bool CheckSelectionVer1(const ObjSelection &objList, bool *pOutAnyCivilUnits, bool *pOutAnyCo
    if (!objList.IsValid())
        return false;
    // local variables:
    int numCivilUnits = 0;
    int numCombat = 0;
    int numAnimating = 0;
    // scan...
    // set values:
    if (pOutAnyCivilUnits)
        *pOutAnyCivilUnits = numCivilUnits > 0;
    if (pOutAnyCombatUnits)
        *pOutAnyCombatUnits = numCombat > 0;
    if (pOutNumAnimating)
        *pOutNumAnimating = numAnimating;
    return true;
}
```

As you can see above, the function uses a lot of output parameters (in the form of raw pointers), and it returns true/false to indicate success (for example the input selection might be invalid). The implementation of the function is not relevant now, but here's an example code that calls this function:

```
int main(){
   ObjSelection sel;

bool anyCivilUnits { false };
   bool anyCombatUnits { false };
   int numAnimating { 0 };

   if (CheckSelectionVer1(sel, &anyCivilUnits, &anyCombatUnits, &numAnimating))
   {
     std::cout << "ok...\n";
   }
}</pre>
```

How can we improve the function?

There might be several things:

- Look at the caller's code: we have to create all the variables that will hold the outputs. It definitely generates code duplication if you call the function in many places.
- Output parameters: Core guidelines suggests not to use them.
 - F.20: For "out" output values, prefer return values to output parameters
- If you have raw pointers you have to check if they are valid. You might get away with the checks if you use references for the output parameters.
- What about extending the function? What if you need to add another output param?

Anything else?

How would you refactor this?

Motivated by Core Guidelines and new C++17 features, here's the plan for how we can improve this code:

1. Refactor output parameters into a tuple that will be returned.

- 2. Refactor tuple into a separate struct and reduce the tuple to a pair.
- 3. Use std::optional to express if the value was computed or not.
- 4. Use std::variant to convey not only the optional result but also the full error information.

That pretty much sums up the Use Case. By now you have observed 'tuples'. Let's see how the Tuple Version works in the next lesson.