Project Documentation:

Most of the code is construction of GUI and handling or storing the inputs.

The code is written in python using the Kivy framework for GUI.

The core function for calculation is the “check” function in “MyManager” class.

```python
def check(self, to_check, scroll_obj):
    def print_q(Q: Queue):
        size_q = Q.qsize()
        for ii in range(size_q):
            temp = Q.get_nowait()
            print(temp, end="," if ii != size_q - 1 else "\n")
            Q.put_nowait(temp)

    if len(to_check.strip()) > 0:
        q = Queue()
        q.put(State(new_state=self.starting, stack_pointer=[self.stack_base]))
        text = f"processing input {self.index!s}:\n"
        self.index += 1
        for s in to_check:
            size = q.qsize()
            text += f"--->input character: {s} :\n"
            for i in range(size):
                text += f"------->Queue update {i} :\n"
                cur: State = q.get_nowait()
                text += f"----------->current is : {cur!s}\n"
                # scroll_obj.parent.text += str(cur) + "\n"
                for r in self.rules:
                    if r['from'] == cur.state and r['with'] == s and r['while'] == cur.stack[-1]:
                        if not (r['new-stack'][-1] != self.stack_base and cur.stack[-1] == self.stack_base):
                            q.put(State(new_state=r['to'], stack_pointer=cur.stack[:-1],
                                        stack_changes=r['new-stack']))
                # print_q(q)

        # scroll_obj.parent.text += "-----------------\n"
        accept = False
        while q.qsize() > 0:
            cur: State = q.get_nowait()
            if cur.state in self.accepted:
                accept = True
                break
        text += f"input acceptance : {accept!s}\n"
        # print(accept)
        text += "-" * 40 + ">ended.\n"
        scroll_obj.parent.text += text
```

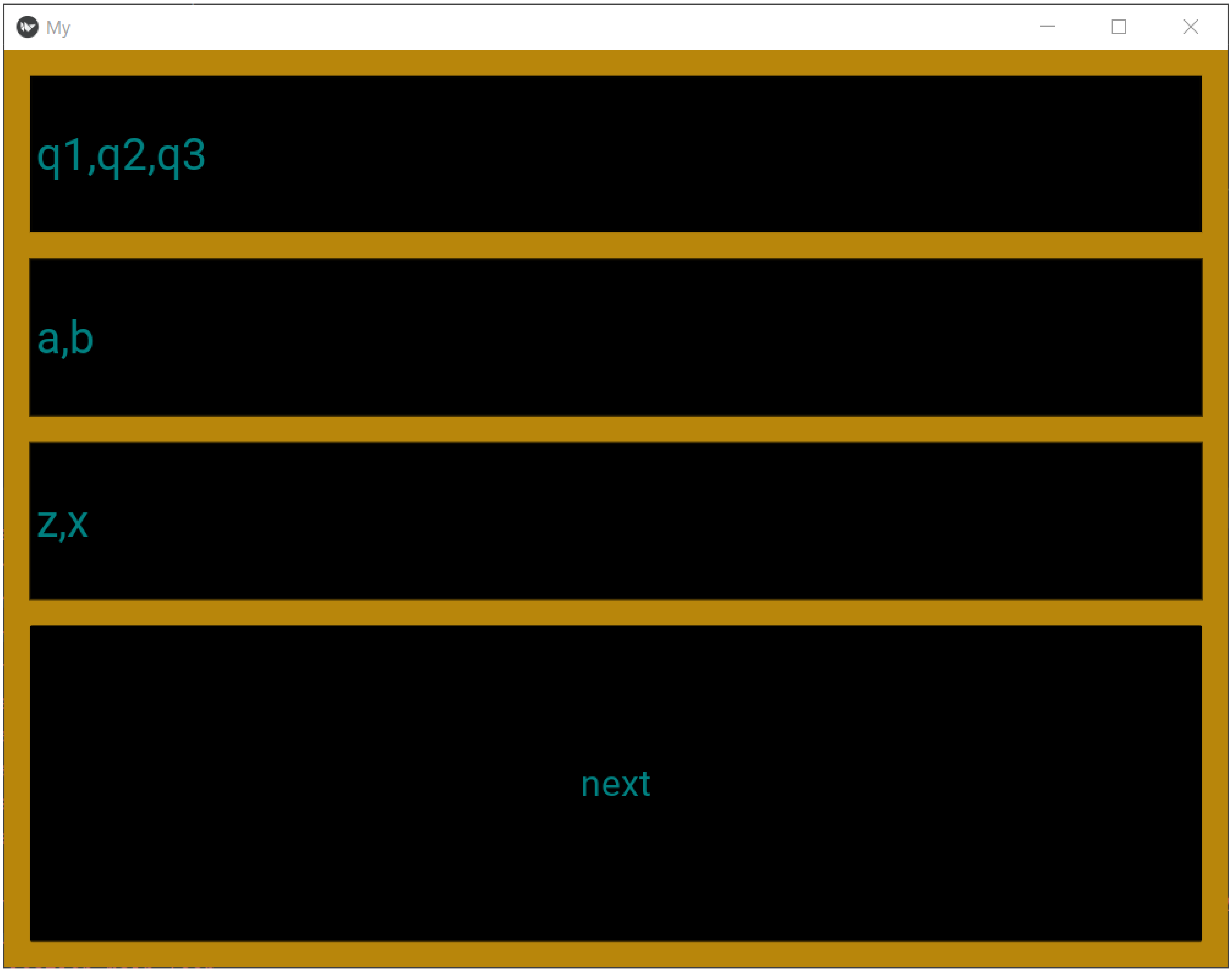
This function uses a queue for storing States that contains current state of machine and elements of stack.

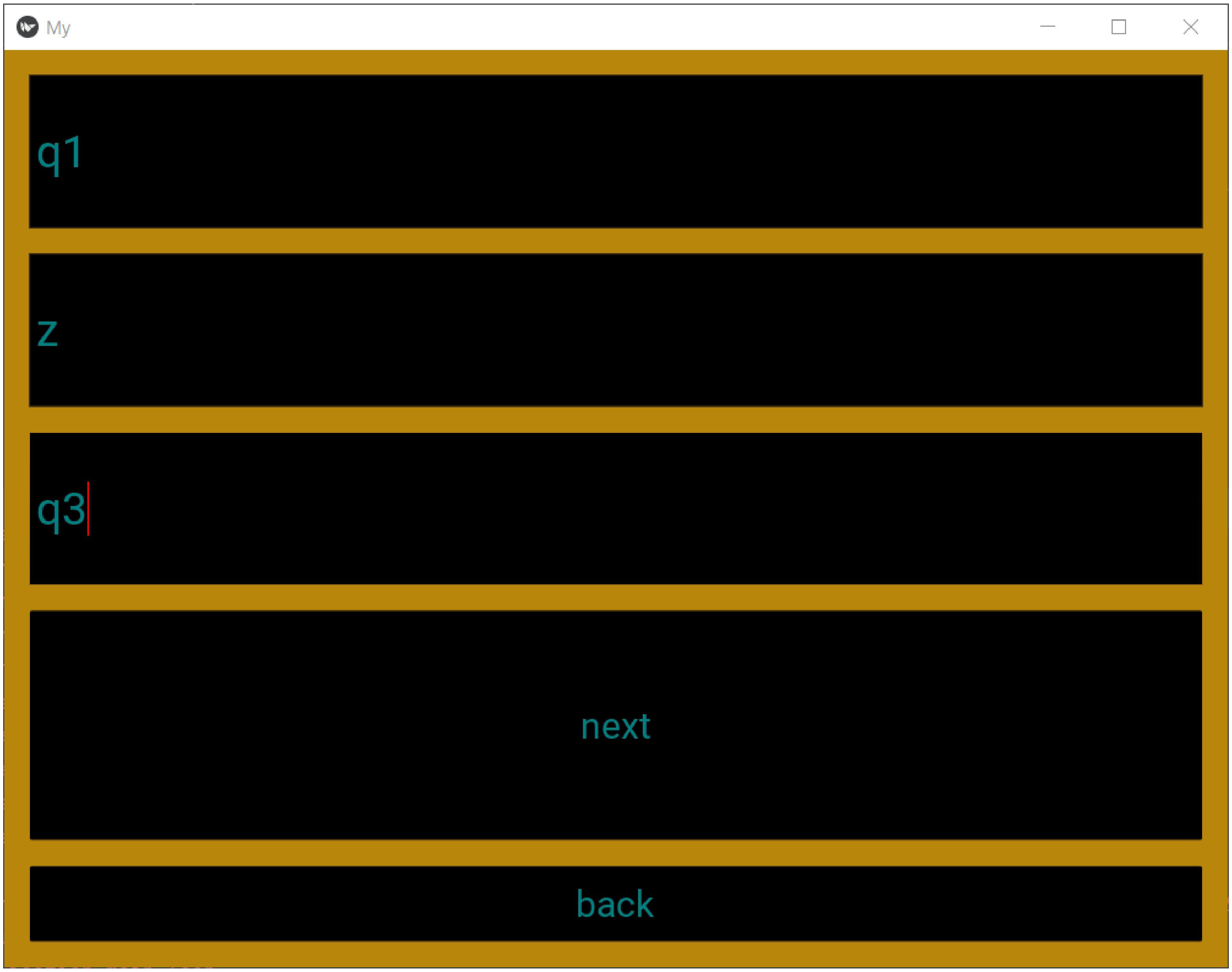
In each iteration with a given input character it will applies all the compatible rules on all the States in the queue.

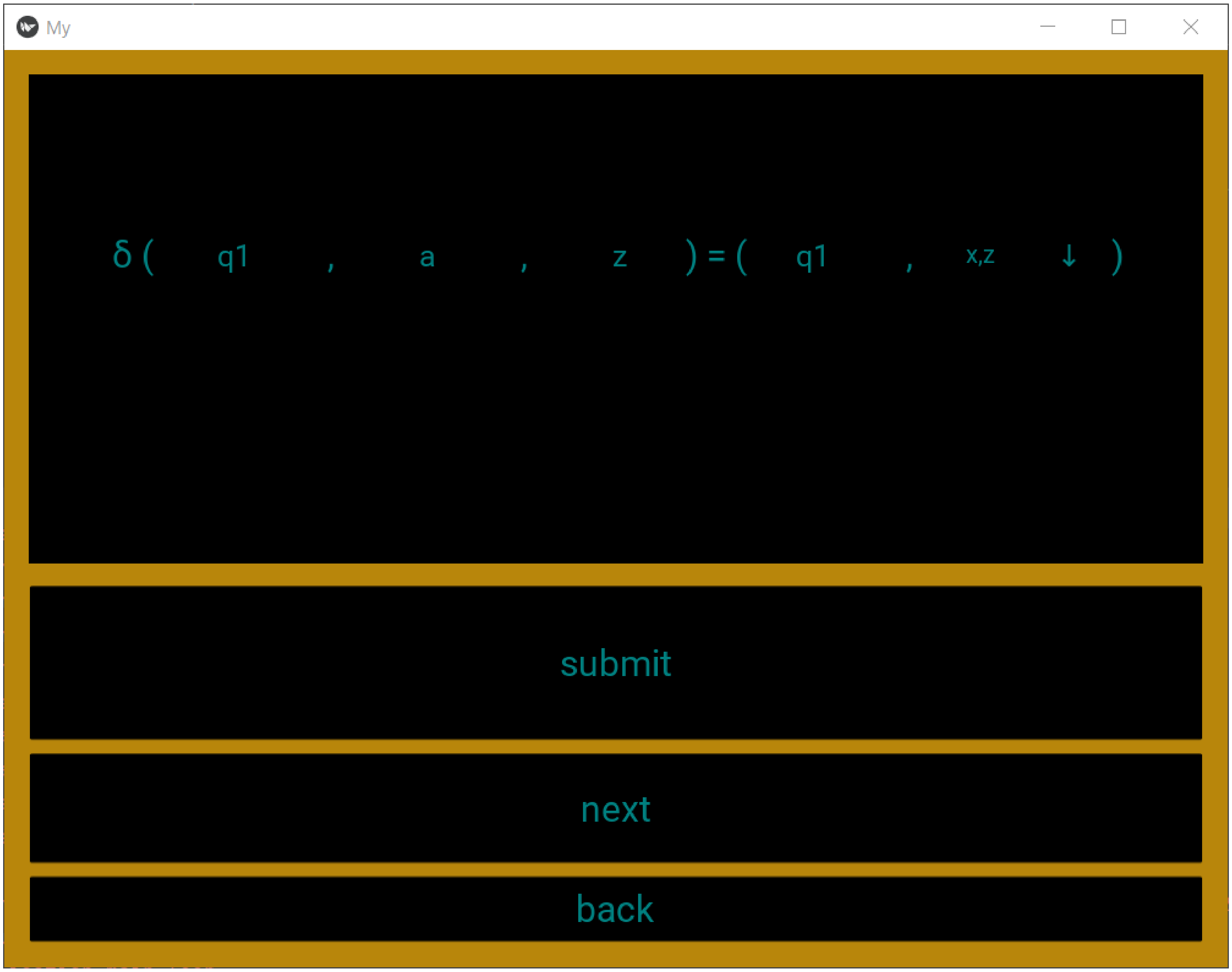
This will fill the queue with new possible States. In the end the queue only stores the reachable States with the given input string and starting state.

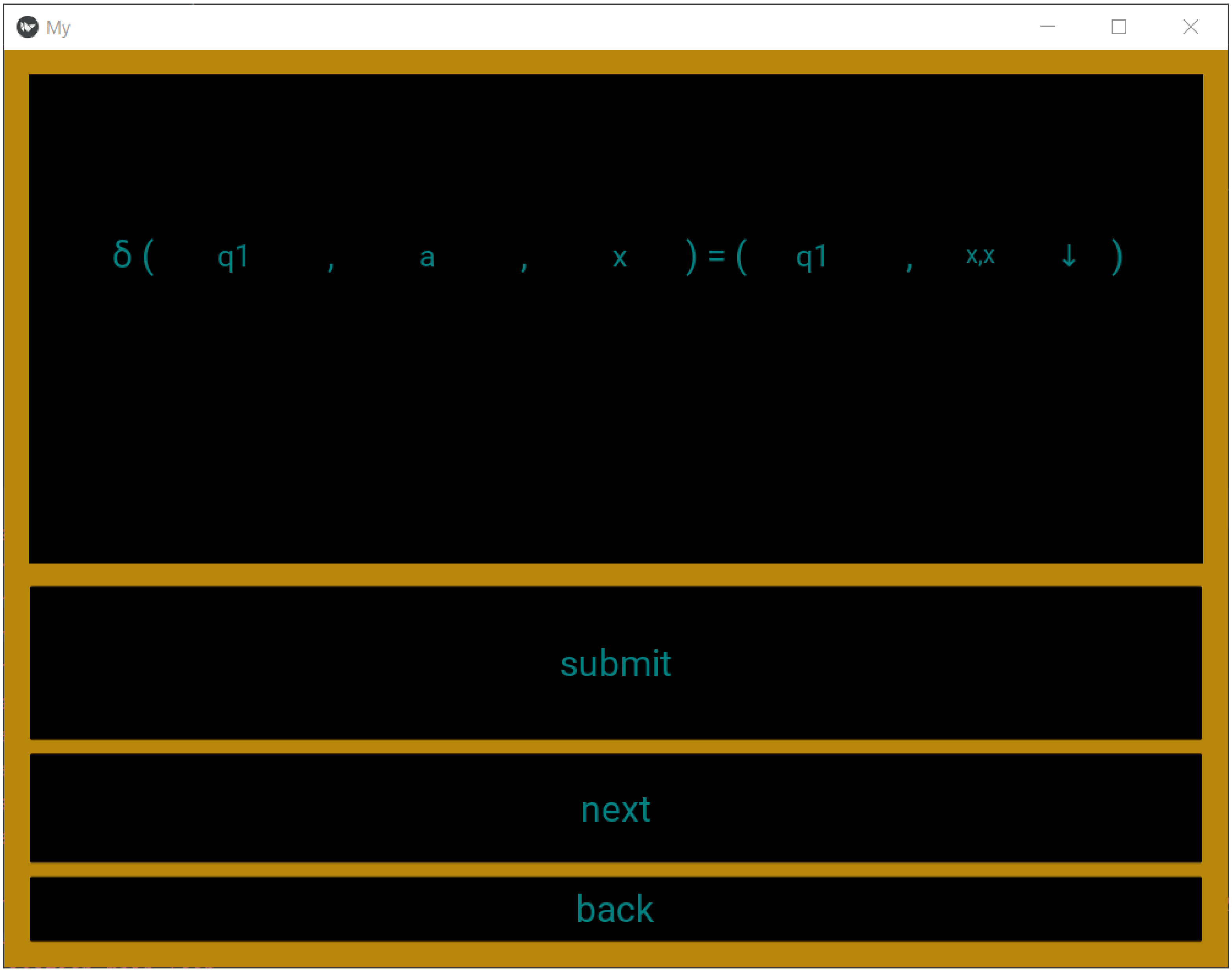
If there is at least one accepting state in the queue after iteration for the input string the string will be accepted.

Here is an input example for the language L = $\{a^n b^{n+1} | \ n >= 1\}$:

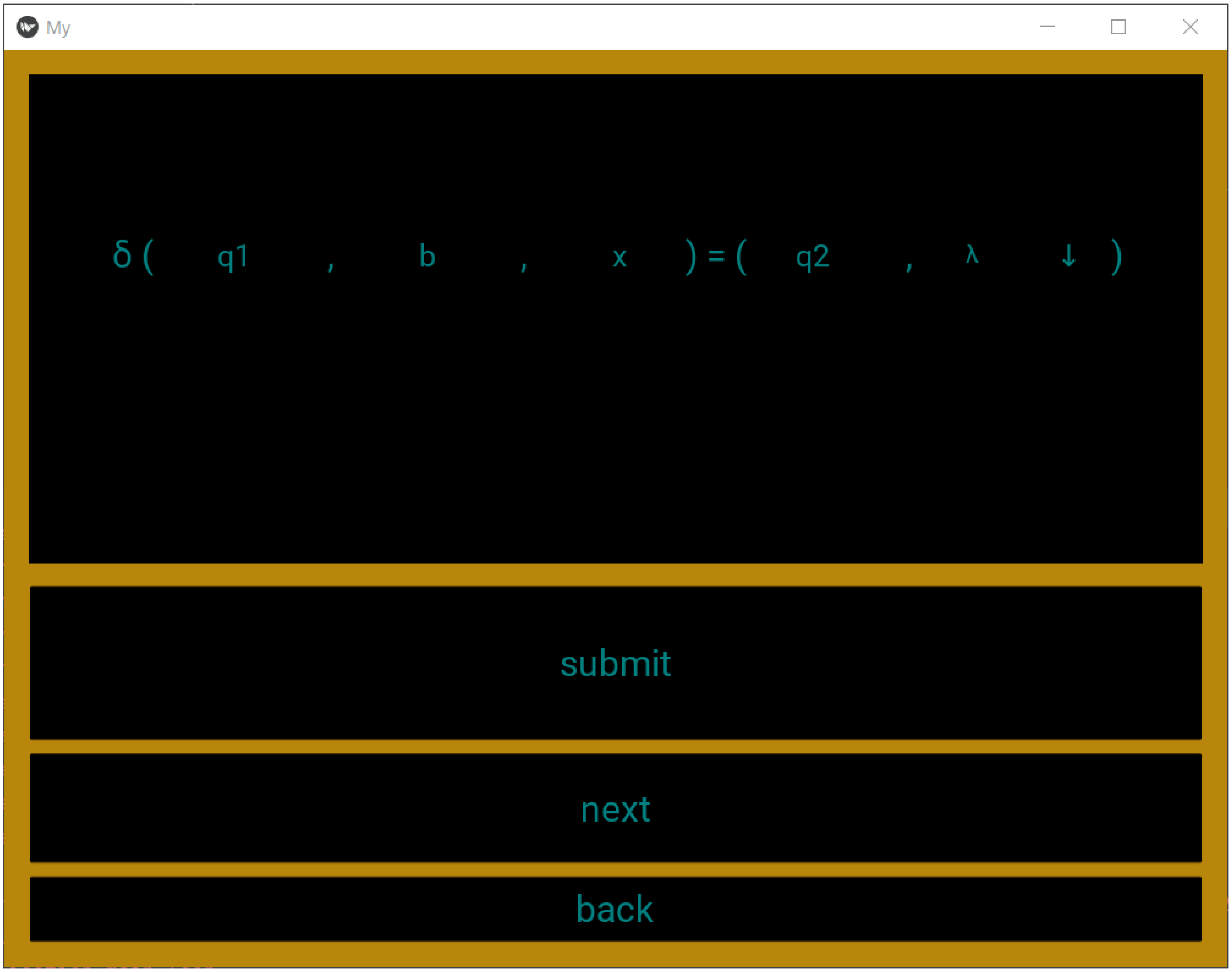

My
q1,q2,q3
a,b
z,x
next

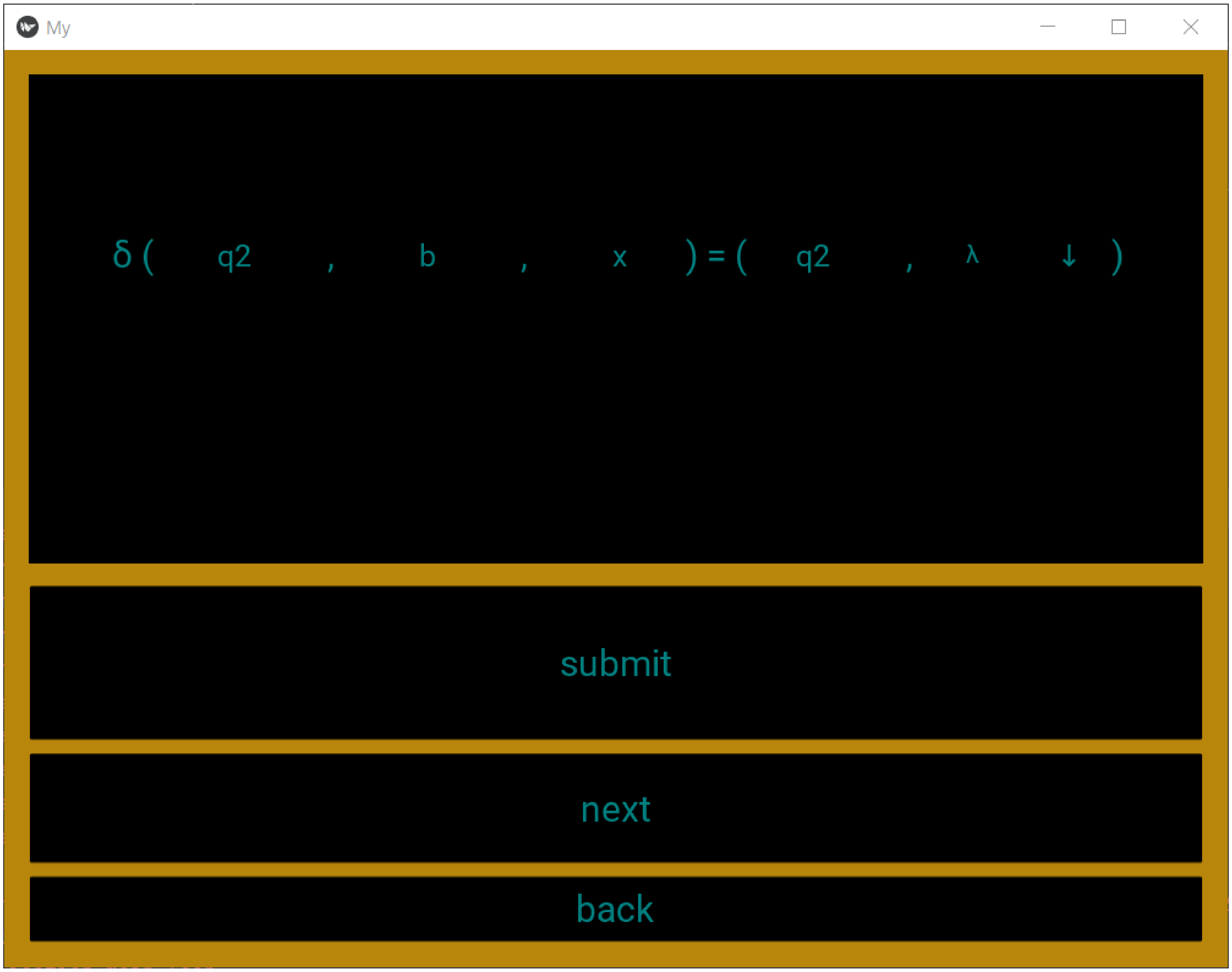

My
q1
z
q3
next
back


My
δ (q1 , a , z) = (q1 , x,z ↓)
submit
next
back

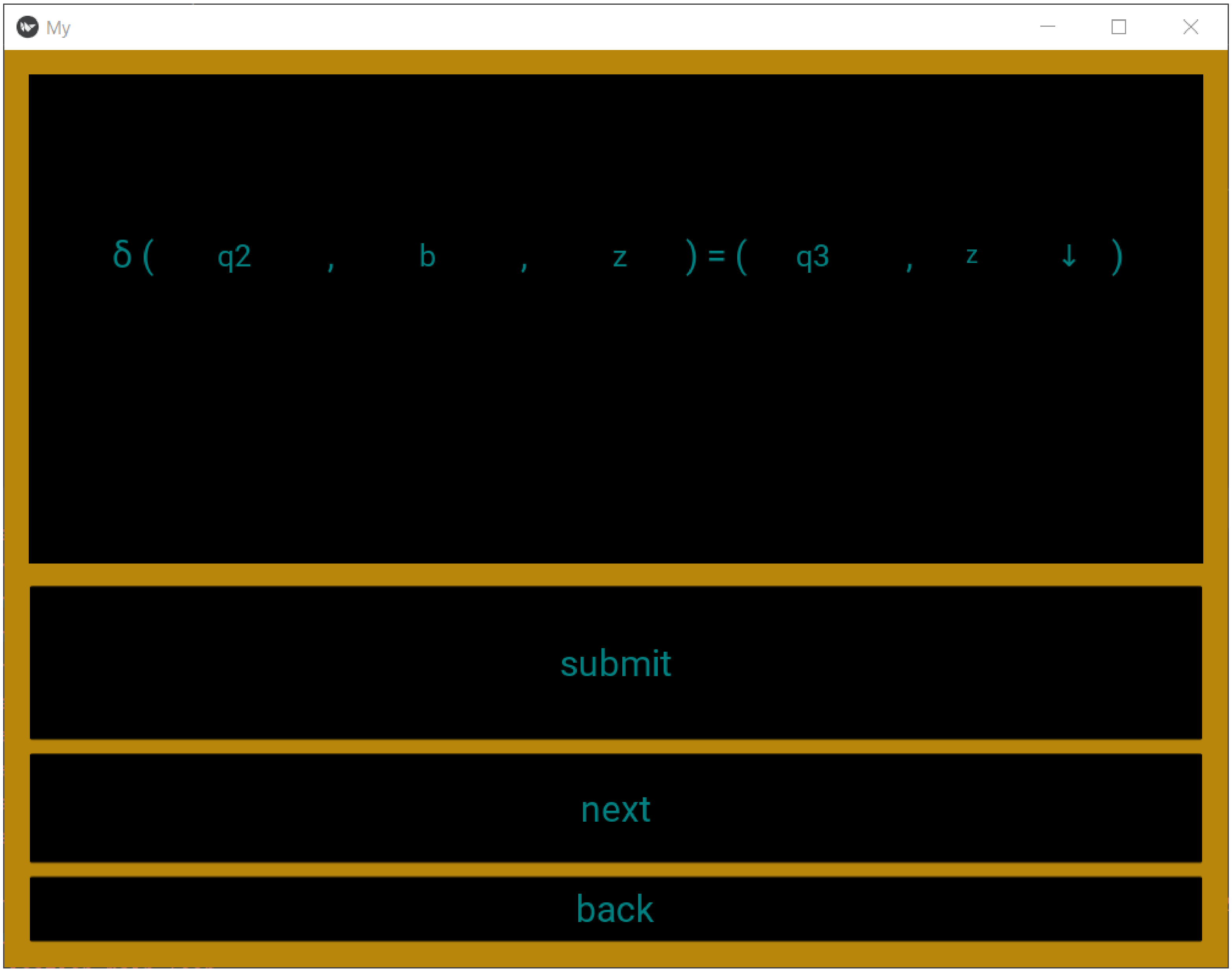


δ (q1 , a , x) = (q1 , x,x ↓)

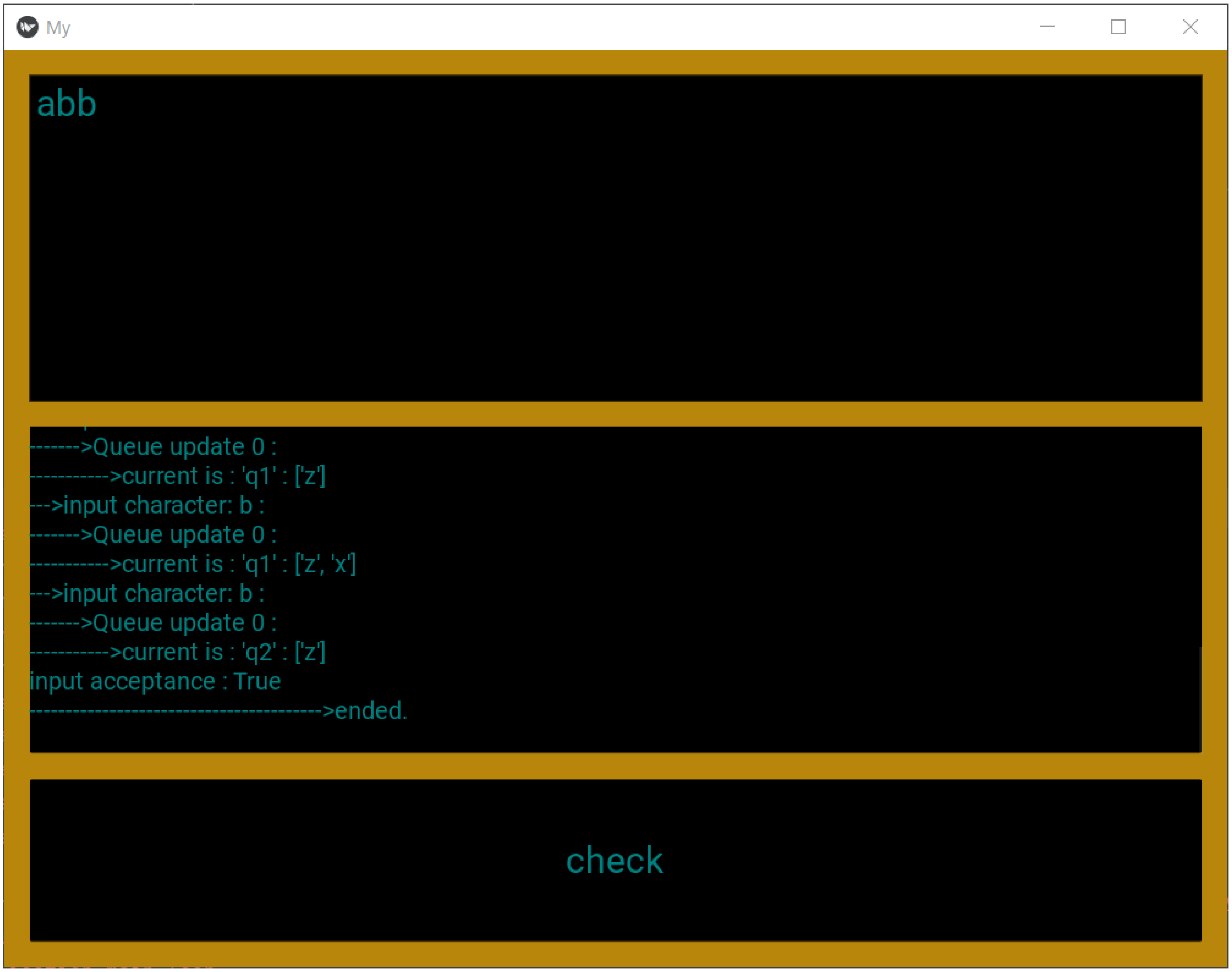

My
δ (q1 , b , x) = (q2 , λ ↓)
submit
next
back



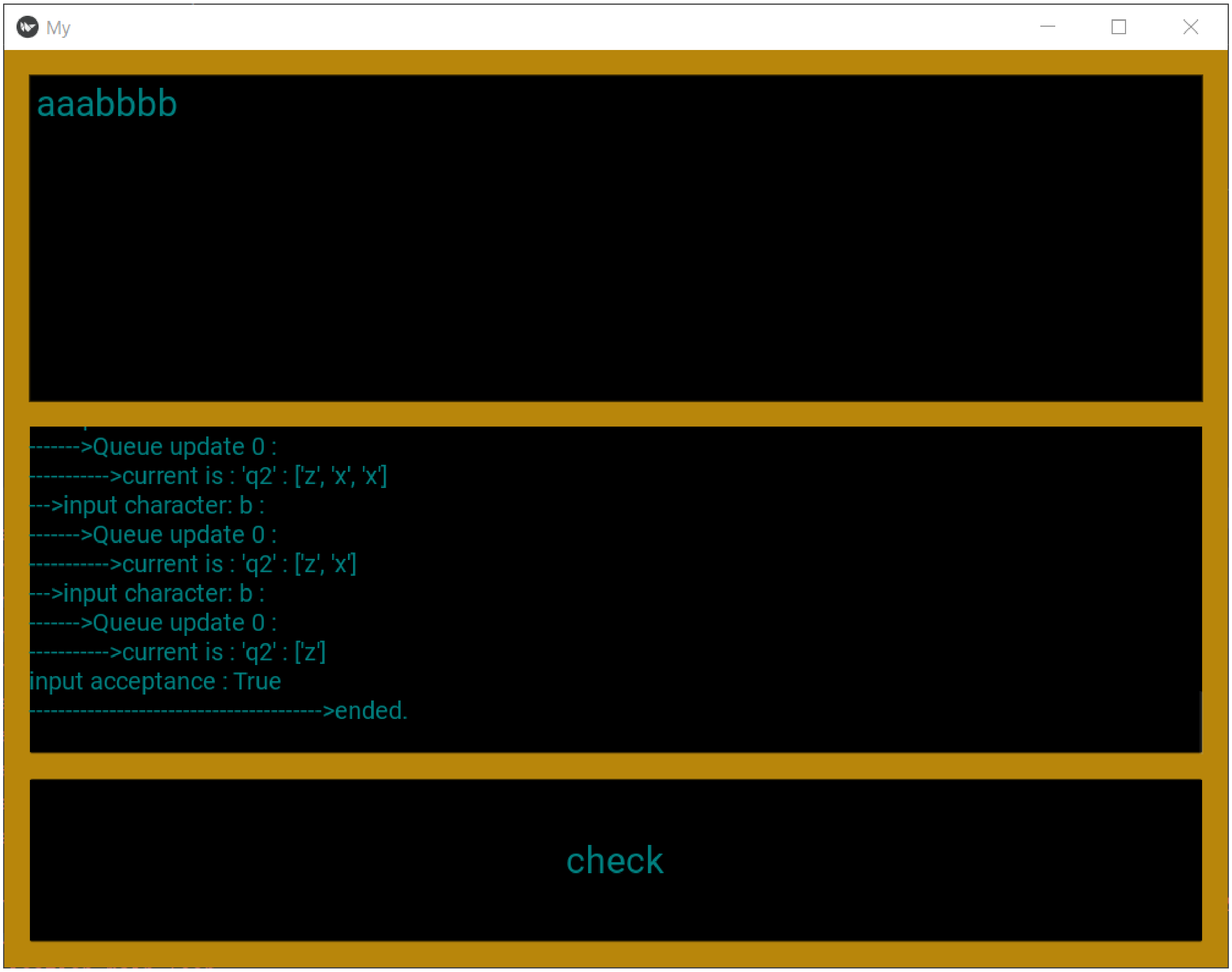
δ (q2 , b , x) = (q2 , λ ↓)



δ (q2 , b , z) = (q3 , z ↓)

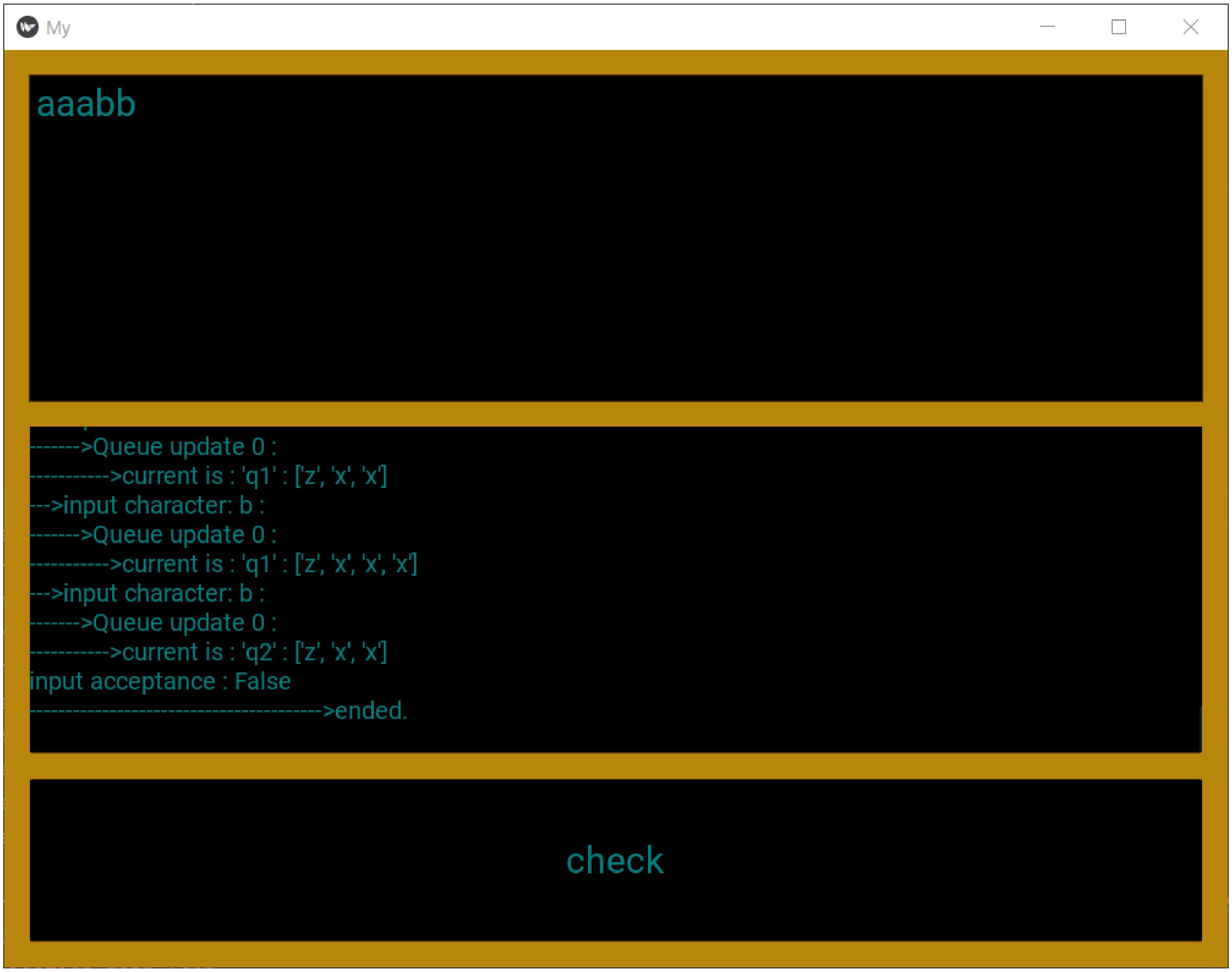


```
------->Queue update 0 :
----------->current is : 'q1' : ['z']
--->input character: b :
------->Queue update 0 :
----------->current is : 'q1' : ['z', 'x']
--->input character: b :
------->Queue update 0 :
----------->current is : 'q2' : ['z']
input acceptance : True
---------------------------------------->ended.
```



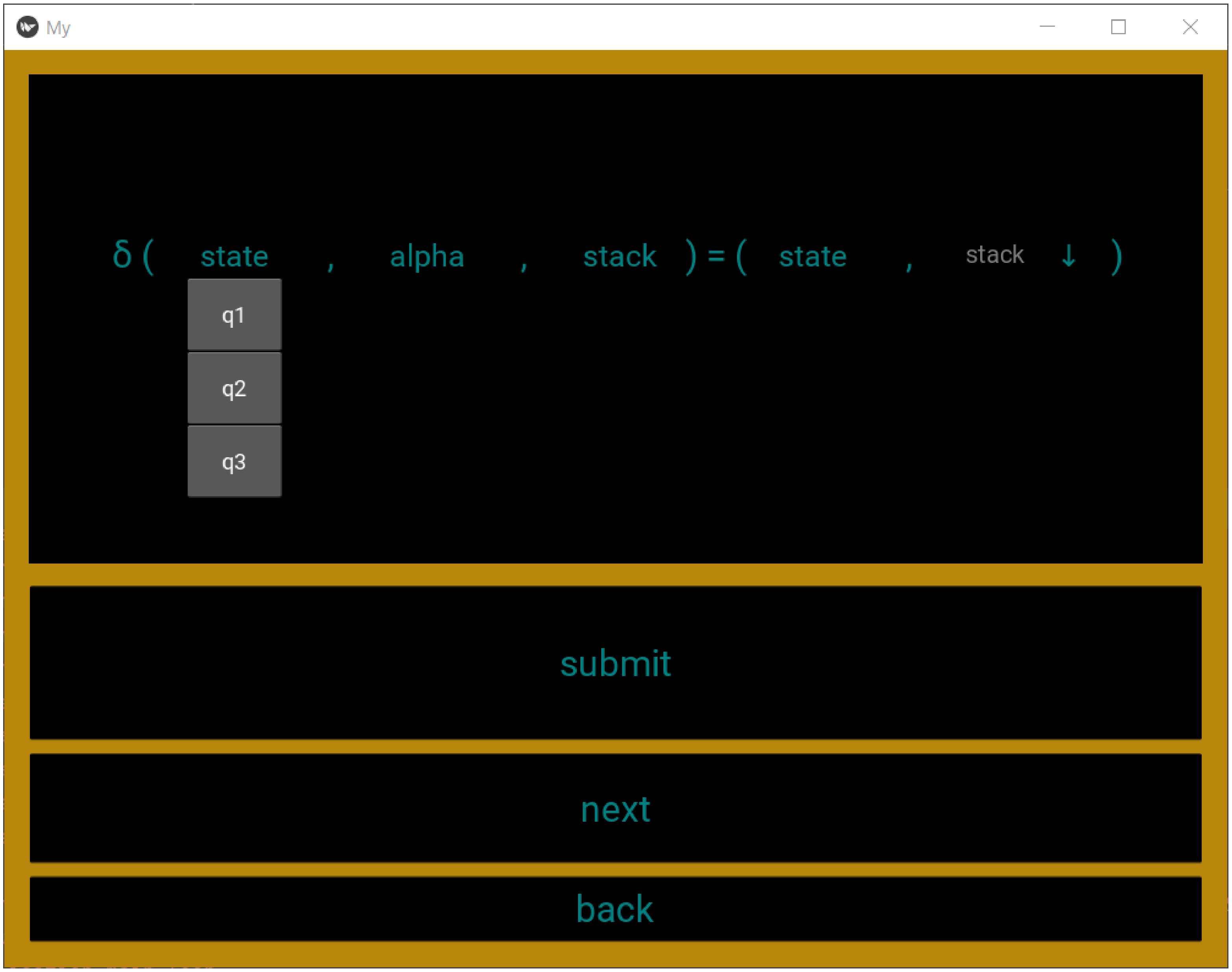
```
aaabbbb
```

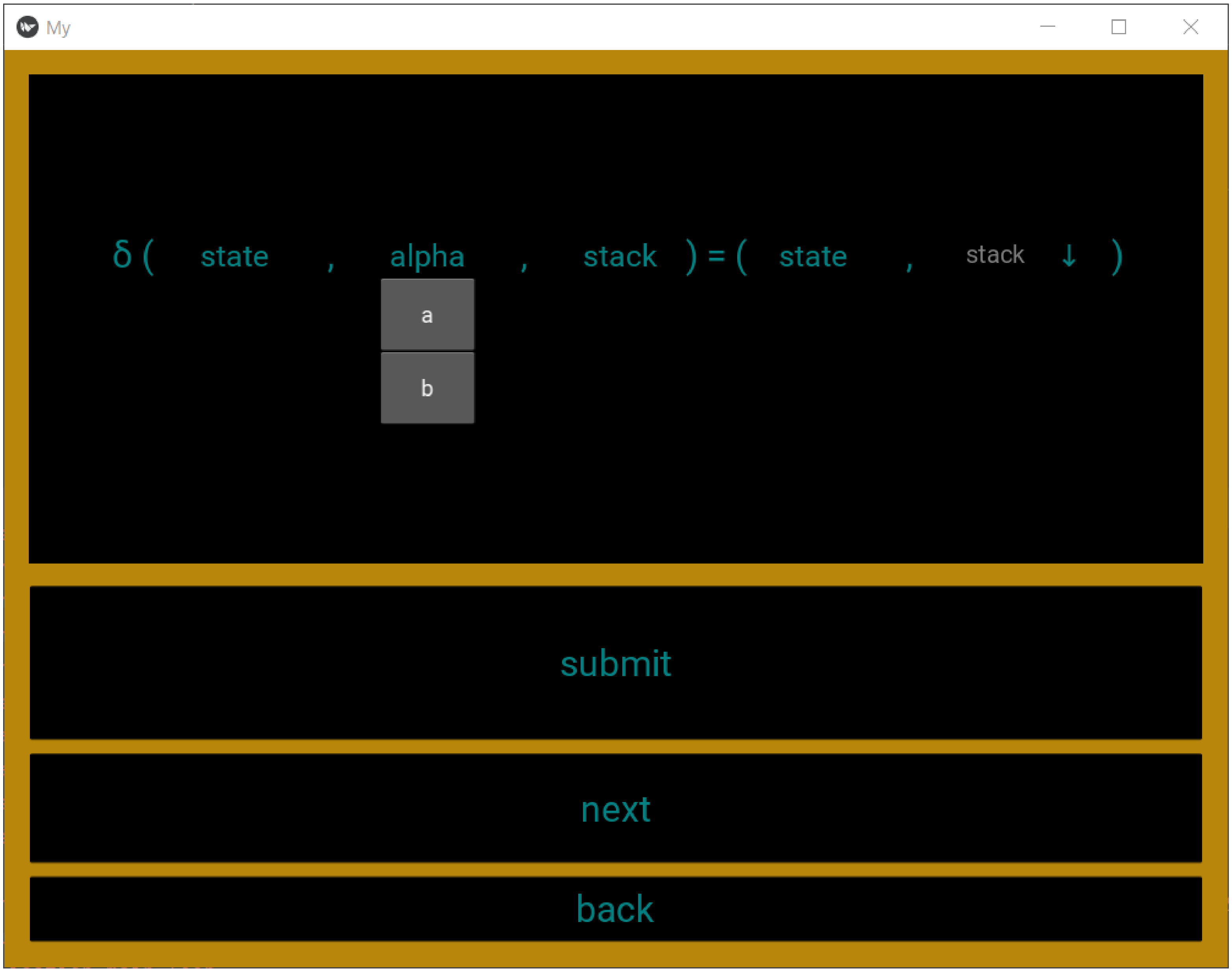
```
------->Queue update 0 :
----------->current is : 'q2' : ['z', 'x', 'x']
--->input character: b :
------->Queue update 0 :
----------->current is : 'q2' : ['z', 'x']
--->input character: b :
------->Queue update 0 :
----------->current is : 'q2' : ['z']
input acceptance : True
---------------------------------------->ended.
```

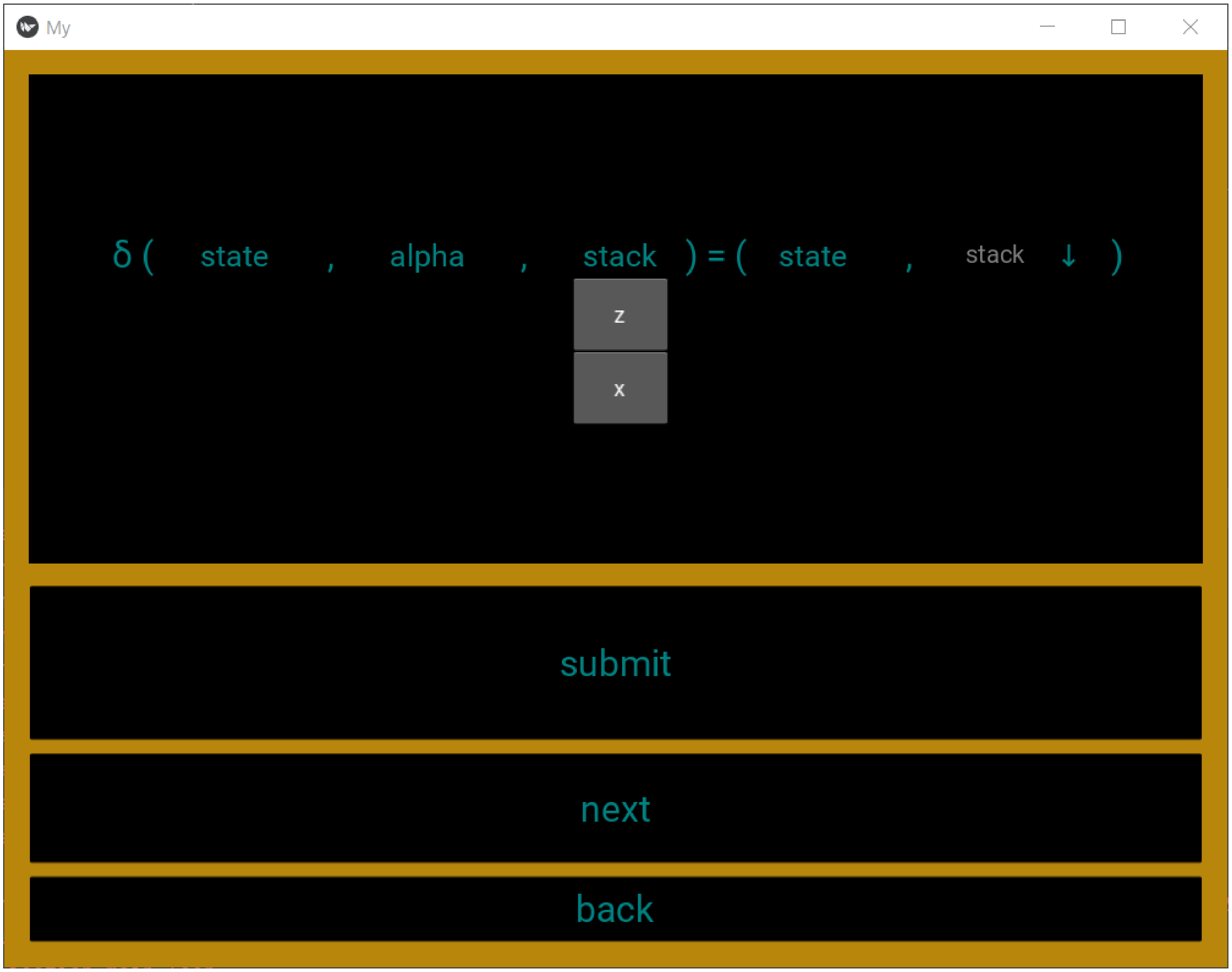


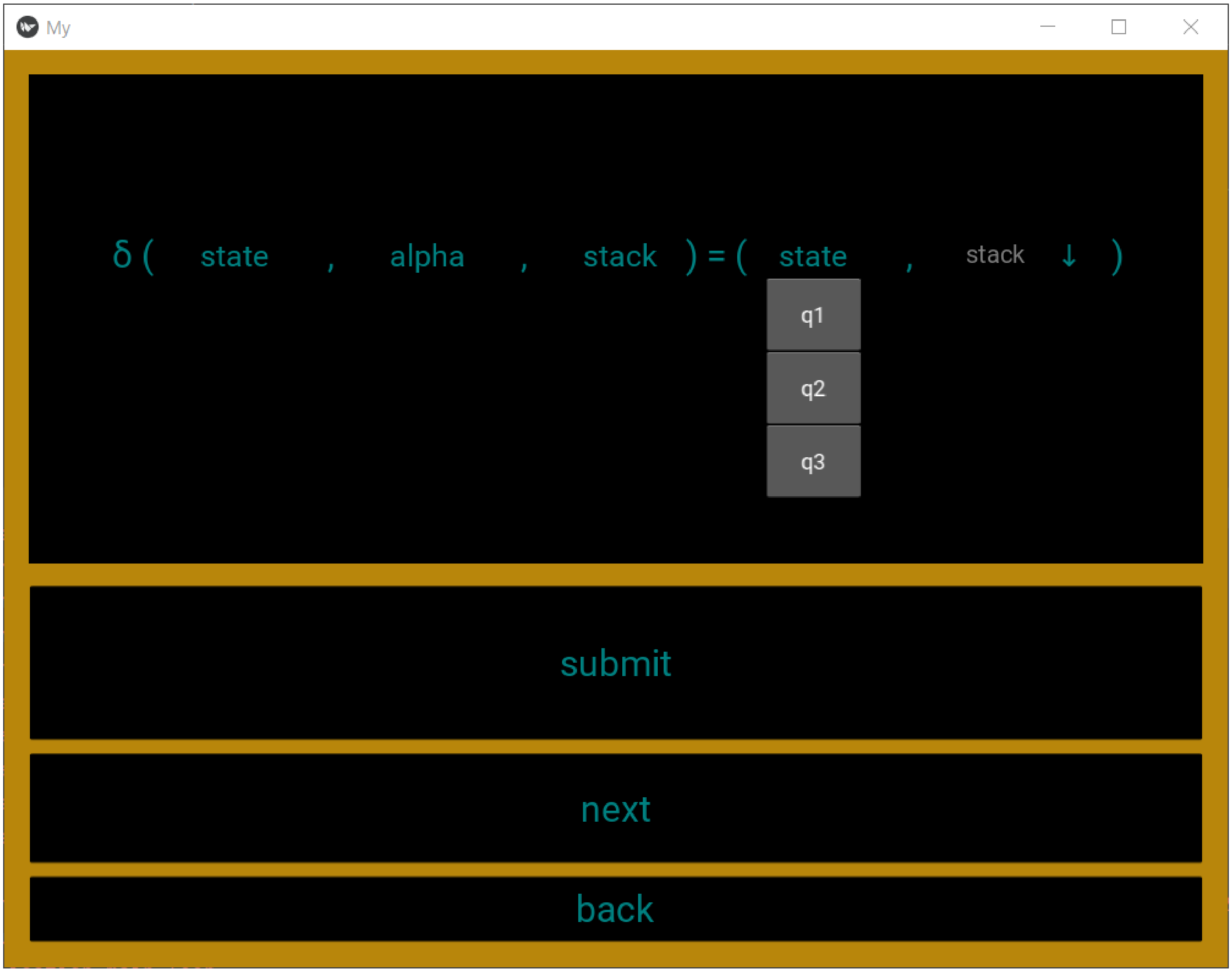
```
------->Queue update 0 :
----------->current is : 'q1' : ['z', 'x', 'x']
--->input character: b :
------->Queue update 0 :
----------->current is : 'q1' : ['z', 'x', 'x', 'x']
--->input character: b :
------->Queue update 0 :
----------->current is : 'q2' : ['z', 'x', 'x']
input acceptance : False
---------------------------------------->ended.
```

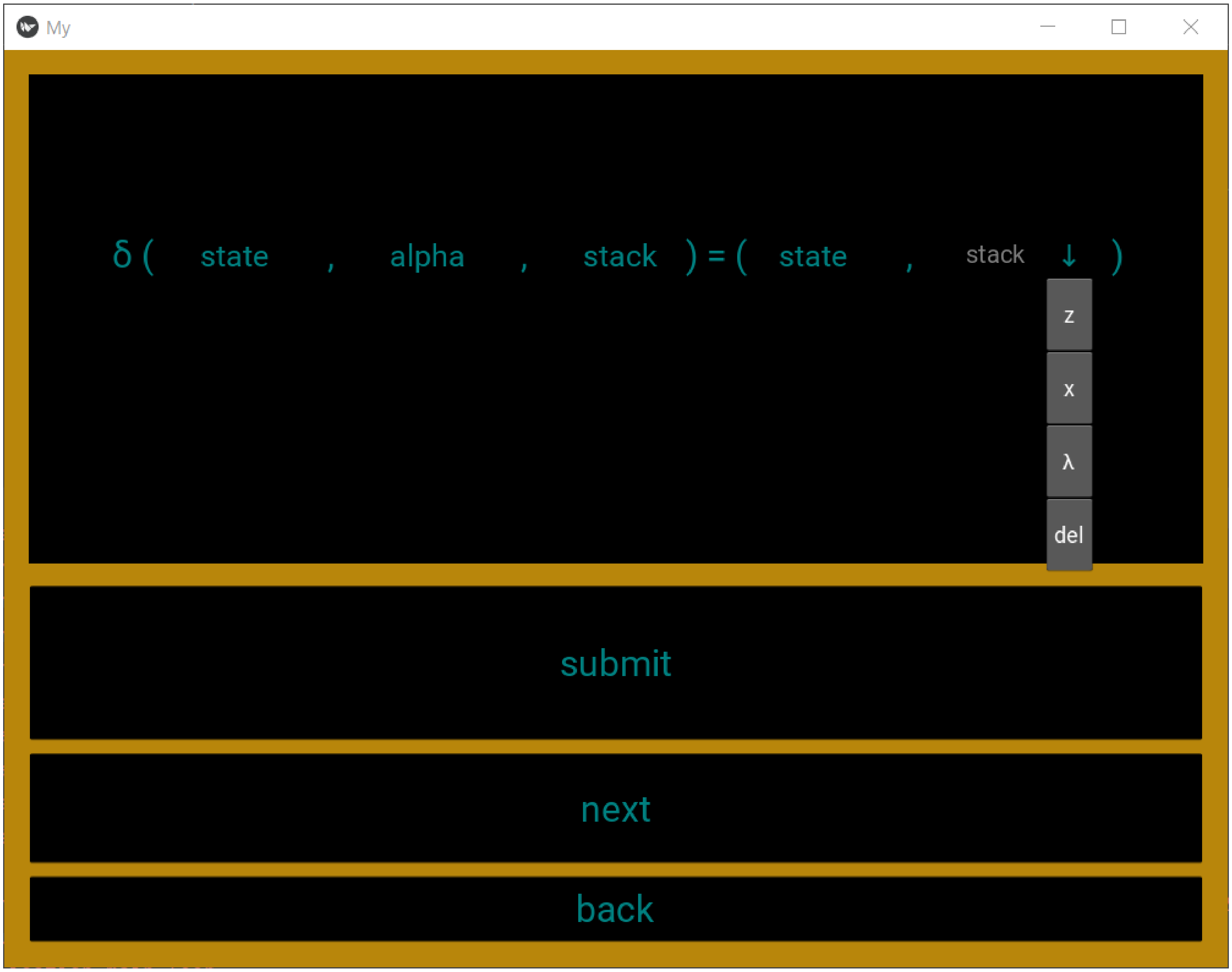
Here is how to use the rule input interface:


My
δ (state , alpha , stack) = (state , stack ↓)
q1
q2
q3
submit
next
back


My
δ (state , alpha , stack) = (state , stack ↓)
a
b
submit
next
back


My
δ (state , alpha , stack) = (state , stack ↓)
z
x
submit
next
back


My
δ (state , alpha , stack) = (state , stack ↓)
q1
q2
q3
submit
next
back


My
δ (state , alpha , stack) = (state , stack ↓)
z
x
λ
del
submit
next
back