feature/rush-cancel Additions

Multiple new features have been made in this branch. The functionality for each will be explained, both on the gameplay side and the coding side.

Features added:

- Rush Cancels
- Screen Freezing
- Speed Trails

Rush Cancels

The Rush Cancel is the big new gameplay feature added in this branch.

There are three versions of the Rush Cancel: a running version, a jumping version, and a falling version.
The running version makes the player run forward about ⅓ of the screen. This is performed if Rush Cancel is inputted on the ground.
The jumping version makes the player do a quick jump. This is performed if Rush Cancel is inputted on the ground and you hold up by the end of the screen freeze.
The falling version makes the player quickly fall to the ground. This is performed if Rush Cancel is inputted in the air.
Activating Rush Cancel costs 50 meter, and all versions of the Rush Cancel are projectile invincible. You can perform Rush Cancels at any time when you are not getting hit.

The Code

Multiple new variables and states have been added for Rush Cancels. These are defined in oPlayerController/Create.

```
// Variables for Rush Cancel
rcActivated = false;
rcBuffer = false; // Used to activate Rush Cancel after exiting screen freeze
rcBufferTimer = 0; // Doesn't activate Rush Cancel if in buffer for more than the designated interval
rcBufferInterval = 0;
rcFreezeTimer = 0; // Counts up to 30 frames, then deactivates the freeze frame
rcForwardTimer = 0; // Handles duration of Rush Cancel Forward run
runButtonPressed = false; // Triggers when the run button is pressed
holdRunButtonTimer = 16; // Determines the amount of frames the run button is held
pressSpecialButtonTimer = 16; // Determines the amount of frames since the special button was pressed
```

```
    RUSH_CANCEL_FORWARD,
    RUSH_CANCEL_UP,
    RUSH_CANCEL_AIR,
```

The process for the Rush Cancel goes by a bit of a step-by-step process in oPlayerController/Step.

1. To activate the Rush Cancel, the game checks for a multitude of things.
    a. The first check is if it is inputted properly. There are two ways to perform the Rush Cancel. The first method is to press the run button and the special button at the same time (within 4 frames of each other). The second method is to double tap dash and press the special button at the same time. The second input of the double tap dash and the special button have to be inputted within 7 frames of each other.
    b. The second check is if the player is not in a state where they are unable to act (i.e. any state where they are getting hit).
    c. The third check is if the player isn't getting it at the moment. There should be no hitstun and no blockstun.
    d. The fourth check is if the player has at least 50 meter, as that is the required amount of meter to perform Rush Cancels.
    e. The fifth and final check is if the player has already activated Rush Cancel and it hasn't completed yet.

```
// If the run button and special button are pressed within 4 frames of each other, activate rush cancel
// Also works with double tap forward, in which case the leniency is 7 frames
if ((((runButton || special) && pressSpecialButtonTimer <= 4 && holdRunButtonTimer <= 4)
    || (pressSpecialButtonTimer <= 7 && holdForwardTimer <= 7 && runningForward && !runButton))
    && state != eState.BEING_GRABBED // Prevent Rush Cancels during any of these states...
    && state != eState.THROW_TECH
    && state != eState.HURT
    && state != eState.LAUNCHED
    && state != eState.KNOCKED_DOWN
    && state != eState.GETUP
    && state != eState.BLOCKING
    && hitstun <= 0 && blockstun <= 0 && !FAvictim // and when the player gets hit...
    && superMeter >= 50 // and if the player doesn't have enough meter...
    && !rcActivated // and if the player already activated Rush Cancel and it hasn't completed yet.
    && !rcBuffer)
{
```

2. Once all of the checks have been completed, the game then determines if the Rush Cancel can happen immediately or if a buffer should be activated.
    a. If the opponent has already performed a screen freeze, a buffer is activated to perform the Rush Cancel after the opponent's screen freeze is finished. The only exception to this is if both players activate Rush Cancels at the exact same time.

```
187     if (opponent != noone && opponent.activateFreeze)
188     {
189         // 1 frame buffer to determine if both players activated RC at the same time
190         if (opponent.rcActivated && opponent.rcFreezeTimer > 1)
191         {
192             rcActivated = true;
193             rcFreezeTimer = 0;
194             superMeter -= 50;
195             projectileInvincible = true;
196             global.hitstop = 0;
197             activateFreeze = true;
198             global.freezeTimer = true;
199             state = eState.SCREEN_FREEZE;
200             instance_create_layer(x, y, "Instances", oRushCancel);
201         }
202         else
203         {
204             rcBuffer = true;
205             rcBufferTimer = 0;
206             rcBufferInterval = 30;
207         }
208     }
```

    b. If the player is grabbing the opponent, activate the buffer to wait for after the grab move is complete.

```
209     else if (opponent != noone && opponent.isGrabbed) // Activates buffer when grabbing
210     {
211         rcBuffer = true;
212         rcBufferTimer = 0;
213         rcBufferInterval = 999;
214     }
```

    c. Otherwise, activate the Rush Cancel immediately.

```
215     else
216     {
217         rcActivated = true;
218         rcFreezeTimer = 0;
219         superMeter -= 50;
220         projectileInvincible = true;
221         global.hitstop = 0;
222         activateFreeze = true;
223         global.freezeTimer = true;
224         state = eState.SCREEN_FREEZE;
225         instance_create_layer(x, y, "Instances", oRushCancel);
226     }
```

    rcActivated allows the game to recognize that the player is performing a Rush Cancel as they enter the SCREEN_FREEZE state and rcFreezeTimer determines how long the player is in the

SCREEN_FREEZE state. The player also becomes projectile invincible once they activate the Rush Cancel so they are invincible to them during the freeze. The importance of lines 221 - 224 will be explained in the Screen Freeze section.

d. This is more like step 2.5. If the buffer is activated, the code goes through a number of if statements outside of the giant Rush Cancel if statement.

```
228 if (rcBufferTimer > rcBufferInterval)
229 {
230     rcBuffer = false;
231     rcBufferTimer = 0;
232 }
233 // Checks for either if the opponent activated screen freeze or if they are being grabbed
234 if (rcBuffer && rcBufferTimer <= rcBufferInterval && opponent != noone && !opponent.activateFreeze
235     && !opponent.isGrabbed && state != eState.HITSTOP)
236 {
237     rcBuffer = false;
238     rcBufferTimer = 0;
239     rcActivated = true;
240     rcFreezeTimer = 0;
241     superMeter -= 50;
242     projectileInvincible = true;
243     global.hitstop = 0;
244     activateFreeze = true;
245     global.freezeTimer = true;
246     state = eState.SCREEN_FREEZE;
247     instance_create_layer(x, y, "Instances", oRushCancel);
248 }
249 if (rcBuffer)
250 {
251     rcBufferTimer++;
252 }
```

i. The first if statement deactivates the buffer if it is inputted too soon during screen freezes. This prevents players from accidentally inputting it during longer bits of screen freezing, such as during supers.

ii. The second if statement checks if the screen isn't already frozen or if the player isn't grabbing the opponent anymore, since those are the reasons the buffer is activated in the first place.

iii. The third if statement simply increments the buffer timer if the buffer is active.

3. Now the player is in the SCREEN_FREEZE state. The following code occurs if the player is performing a Rush Cancel.

```
654     if (rcActivated)
655     {
656         // Screen freeze for Rush Cancel lasts for 30 frames
657         if (rcFreezeTimer >= 30)
658         {
659             rcActivated = false;
660             rcFreezeTimer = 0;
661             activateFreeze = false;
662             global.freezeTimer = false;
663             animTimer = 0; // Reset the animation timer when entering Rush Cancel state
664             speedTrailTimer = 0;
665             comboScaling += 3.0;
666             if (!grounded)
667             {
668                 state = eState.RUSH_CANCEL_AIR;
669             }
670             else if (verticalMoveDir == 1)
671             {
672                 vsp = -global.rcUpSpeed;
673                 jumpHsp = walkSpeed * 1.5 * image_xscale;
674                 state = eState.RUSH_CANCEL_UP;
675                 grounded = false;
676             }
677             else
678             {
679                 rcForwardTimer = 0;
680                 state = eState.RUSH_CANCEL_FORWARD;
681             }
682         }
683         else
684         {
685             rcFreezeTimer++;
686         }
687     }
```

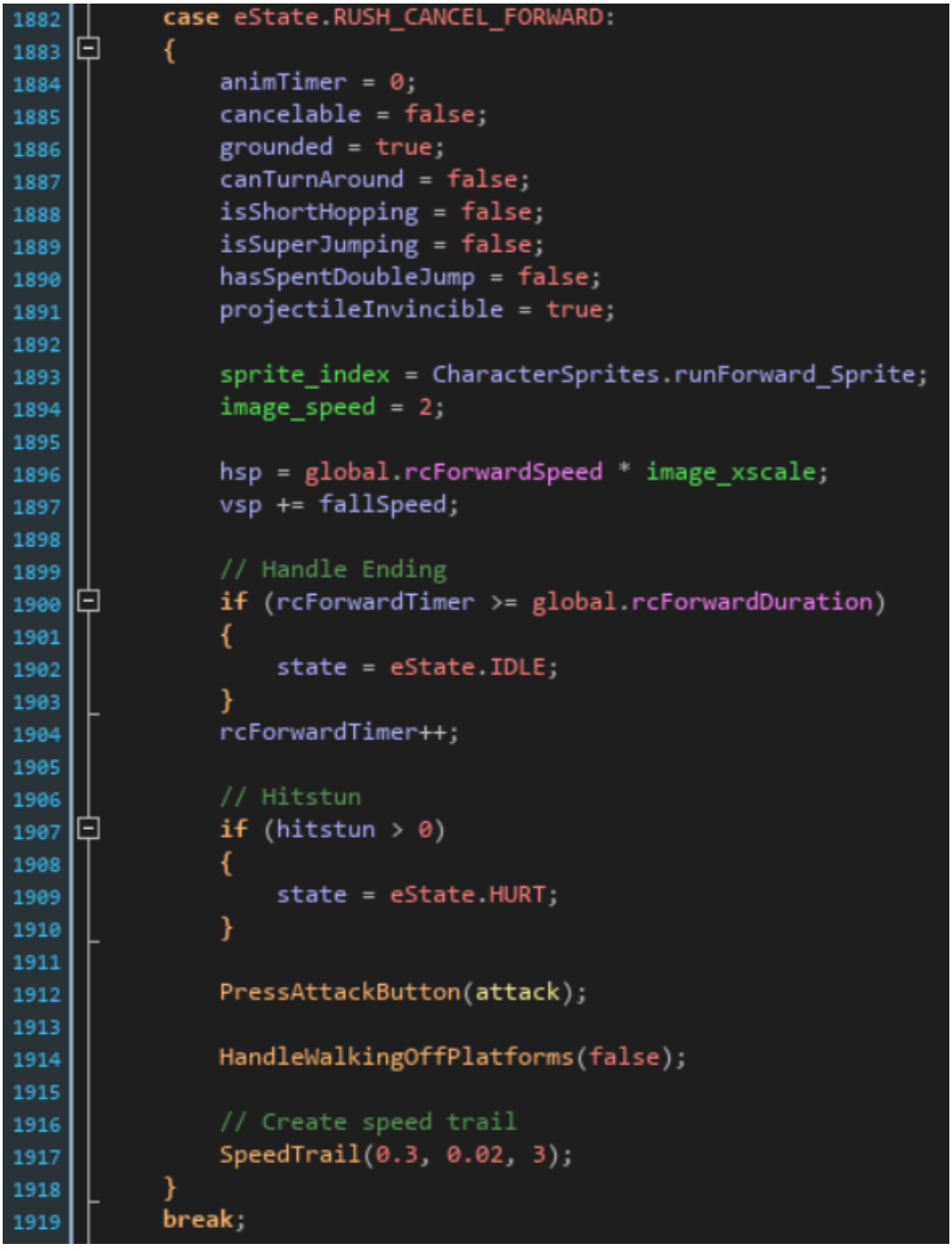
The timer increments during the screen freeze. Once it reaches over 30 frames, rcActivated is set to false, the timer is reset, and the screen freeze is deactivated. From there, the game determines which Rush Cancel state to go to.

a. If the player is in the air, go to RUSH_CANCEL_AIR.
b. If the player is on the ground and they are holding up, go to RUSH_CANCEL_UP. An upward force is applied beforehand since it is basically a jump.
c. Otherwise, go to RUSH_CANCEL_FORWARD. rcForwardTimer is a timer to determine how long the player runs for in this state.

4. The final step in the Rush Cancel process is what happens in each of the Rush Cancel states. Each state uses global movement variables that can be found in oGlobalVars/Create.

```
92  // Rush Cancel global variables (both players use them)
93  global.rcForwardSpeed = 4;
94  global.rcForwardDuration = 12;
95  global.rcUpSpeed = 5;
96  global.rcUpForwardSpeed = 2;
97  global.rcUpFallSpeed = 0.4;
98  global.rcAirSpeed = 10;
99  global.rcAirHorizontalSpeed = 2;
```

a. RUSH_CANCEL_FORWARD is basically a combination of the RUN_FORWARD and RUN_BACKWARD states. The player moves in a similar way to RUN_FORWARD, but it also has a duration and it can't be interrupted by other forms of movement like in RUN_BACKWARD.



```
1882        case eState.RUSH_CANCEL_FORWARD:
1883        {
1884            animTimer = 0;
1885            cancelable = false;
1886            grounded = true;
1887            canTurnAround = false;
1888            isShortHopping = false;
1889            isSuperJumping = false;
1890            hasSpentDoubleJump = false;
1891            projectileInvincible = true;
1892
1893            sprite_index = CharacterSprites.runForward_Sprite;
1894            image_speed = 2;
1895
1896            hsp = global.rcForwardSpeed * image_xscale;
1897            vsp += fallSpeed;
1898
1899            // Handle Ending
1900            if (rcForwardTimer >= global.rcForwardDuration)
1901            {
1902                state = eState.IDLE;
1903            }
1904            rcForwardTimer++;
1905
1906            // Hitstun
1907            if (hitstun > 0)
1908            {
1909                state = eState.HURT;
1910            }
1911
1912            PressAttackButton(attack);
1913
1914            HandleWalkingOffPlatforms(false);
1915
1916            // Create speed trail
1917            SpeedTrail(0.3, 0.02, 3);
1918        }
1919        break;
```

b. RUSH_CANCEL_UP is basically a super simplified form of the JUMP state, since there is only one speed to control the jump arc.

```
1921        case eState.RUSH_CANCEL_UP:
1922        {
1923            animTimer = 0;
1924            cancelable = false;
1925            sprite_index = CharacterSprites.jump_Sprite;
1926            image_speed = 1;
1927            grounded = false;
1928            canTurnAround = false;
1929            projectileInvincible = true;
1930
1931            vsp += global.rcUpFallSpeed;
1932            hsp = walkSpeed * 1.5 * image_xscale;
1933
1934            PressAttackButton(attack);
1935
1936            // Create speed trail
1937            SpeedTrail(0.3, 0.02, 3);
1938        }
1939        break;
```

c. RUSH_CANCEL_AIR is extremely similar to RUSH_CANCEL_UP, but the movement is different.

```
1941        case eState.RUSH_CANCEL_AIR:
1942        {
1943            animTimer = 0;
1944            cancelable = false;
1945            sprite_index = CharacterSprites.jump_Sprite;
1946            image_speed = 2;
1947            grounded = false;
1948            canTurnAround = false;
1949            projectileInvincible = true;
1950
1951            vsp = global.rcAirSpeed;
1952            hsp = global.rcAirHorizontalSpeed * image_xscale;
1953
1954            PressAttackButton(attack);
1955
1956            // Create speed trail
1957            SpeedTrail(0.3, 0.02, 1);
1958        }
1959        break;
```

5. Once the player exits these states, the entire Rush Cancel is complete.

Screen Freezing

Screen Freezing is a new feature created alongside the Rush Cancels. It is a new system that's implemented in a way so it can be used for different scenarios outside of Rush Cancels.

WARNING: The screen freezing system is extremely delicate. All variables need to be used carefully, or else the system will break.

There are multiple new variables created for screen freezing and a new state is added called SCREEN_FREEZE. These are created in oPlayerController/Create.

```
105 // Screen Freeze variables
106 activateFreeze = false; // Determines if opponent activated screen freeze
107 stateBeforeFreeze = 0; // Different from prevState
108 // Stores movement before SCREEN_FREEZE
109 freezeHSP = 0; // Horizontal speed
110 freezeEnvironmentDisplacement = 0;
111 freezeVSP = 0; // Vertical speed
```

In order to activate the screen freeze, you need to set multiple different variables. The Rush Cancel is shown here as an example.

```
215     else
216     {
217         rcActivated = true;
218         rcFreezeTimer = 0;
219         superMeter -= 50;
220         projectileInvincible = true;
221         global.hitstop = 0;
222         activateFreeze = true;
223         global.freezeTimer = true;
224         state = eState.SCREEN_FREEZE;
225         instance_create_layer(x, y, "Instances", oRushCancel);
226     }
```

Lines 221 - 223 are needed to activate screen freezing, while line 224 is technically optional.

1. global.hitstop needs to be set to 0. If the players are still in hitstop during the screen freeze, this can cause all sorts of weird stuff to happen. The player activating the Rush Cancel doesn't enter the SCREEN_FREEZE state, allowing them to move around while the screen is frozen, and the opponent is stuck in SCREEN_FREEZE forever. This could probably be implemented better, but this method has minimal issues for now.
2. activateFreeze needs to be set to true. There are multiple places in the code where both players read this variable and if either player has activated it. This is mostly used to force the opponent to freeze in place.

3. A new global variable called freezeTimer was created in oGameManager/Create. This is primarily used to freeze the round timer during the screen freeze, but it is also used in other places outside of oPlayerController, such as oProjectileBase to freeze it in place. I might create a separate variable for those cases later down the line in case we want to only freeze the timer and nothing else.
4. In most cases, you would also need to set the player's state to SCREEN_FREEZE upon activating it so the player is frozen. However, you can technically omit this if you don't want the player to be frozen during the screen freeze. This can be used in this case for supers and time stop moves for example.

When the player is in the SCREEN_FREEZE state, they don't animate.

```
509  // Animation pauses during hitstop and when the screen freezes
510  if (global.hitstop == 0 && state != eState.SCREEN_FREEZE)
511  {
512      animTimer++;
513  }
514  else if (global.hitstop != 0)
515  {
516      state = eState.HITSTOP;
517  }
```

Their movement is also completely halted.

```
645  // Handle freezing screen
646  if (state == eState.SCREEN_FREEZE)
647  {
648      image_speed = 0;
649      hsp = 0;
650      environmentDisplacement = 0;
651      vsp = 0;
```

If the opponent has activated screen freeze, the player gets frozen. The if statement that reads this is located under the state == eState.SCREEN_FREEZE if statement.

```
698  // Freezes the player if the opponent freezes the screen
699  if (opponent != noone && opponent.activateFreeze && state != eState.SCREEN_FREEZE)
700  {
701      // Change some states when opponent activates screen freeze
702      if (state == eState.BEING_GRABBED)
703      {
704          state = eState.LAUNCHED;
705          vsp = -3; // Pops the player up a little
706      }
707      // Store all important variables before freezing
708      stateBeforeFreeze = state;
709      freezeHSP = hsp;
710      freezeEnvironmentDisplacement = environmentDisplacement;
711      freezeVSP = vsp;
712      state = eState.SCREEN_FREEZE;
713  }
```

If the player's current state is SCREEN_FREEZE because the opponent activated it, the following if statement located within the state == eState.SCREEN_FREEZE if statement is run to exit the state once screen freeze is deactivated. The player's previous state and movement before the freeze is restored.

```
688     // If opponent froze the screen
689     else if (opponent != noone && !opponent.activateFreeze && !activateFreeze)
690     {
691         hsp = freezeHSP;
692         environmentDisplacement = freezeEnvironmentDisplacement;
693         vsp = freezeVSP;
694         state = stateBeforeFreeze;
695     }
```

If you are going to create a feature that will freeze the screen, you need to create two variables for that feature to make the screen freeze work properly: a boolean to determine if the feature is being used, and an int timer to determine how long the screen freeze will last for. Structure it in the same way as with the Rush Cancel.

```
654     if (rcActivated)
655     {
656         // Screen freeze for Rush Cancel lasts for 30 frames
657         if (rcFreezeTimer >= 30)
658         {
659             rcActivated = false;
660             rcFreezeTimer = 0;
661             activateFreeze = false;
662             global.freezeTimer = false;
663             animTimer = 0; // Reset the animation timer when entering Rush Cancel state
664             speedTrailTimer = 0;
665             comboScaling += 3.0;
666             if (!grounded)
667             {
668                 state = eState.RUSH_CANCEL_AIR;
669             }
670             else if (verticalMoveDir == 1)
671             {
672                 vsp = -global.rcUpSpeed;
673                 jumpHsp = walkSpeed * 1.5 * image_xscale;
674                 state = eState.RUSH_CANCEL_UP;
675                 grounded = false;
676             }
677             else
678             {
679                 rcForwardTimer = 0;
680                 state = eState.RUSH_CANCEL_FORWARD;
681             }
682         }
683         else
684         {
685             rcFreezeTimer++;
686         }
687     }
```

You can place this if statement inside or outside of the SCREEN_FREEZE if statement depending on if you want the player to be frozen, but you must make sure to deactivate the boolean, reset the timer, set activateFreeze to false, and set global.freezeTimer to false.

Speed Trails

Speed trails are a small but aesthetically pleasing feature that makes the game look much cooler. The system itself is very simple and easy to implement.

A new variable is created in oPlayerController called speedTrailTimer and a new object was created called oSpeedTrail. oSpeedTrail basically takes a copy of the current sprite for the player and then fades it over time. It even copies the player's color palette.

To create a speed trail sprite, you have to call a script called SpeedTrail.

```
function SpeedTrail(setStartingOpacity, setFadeSpeed, interval)
{
    if (speedTrailTimer >= interval)
    {
        speedTrailTimer = 0;
        object_set_sprite(oSpeedTrail, sprite_index);
        var instance = instance_create_layer(x, y, "Instances", oSpeedTrail);
        var this = object_index;
        var thisCharacter = selectedCharacter;
        var opponentCharacter = opponent.selectedCharacter;
        with (instance)
        {
            image_index = this.image_index;
            // For some reason, the player 2 sprite is flipped during mirror matches
            if (thisCharacter == opponentCharacter && this.playerID == 2)
            {
                image_xscale = -this.image_xscale;
            }
            else
            {
                image_xscale = this.image_xscale;
            }

            if (this.playerID == 1)
            {
                PaletteSetup(global.p1PaletteID, this.selectedCharacter);
            }
            else
            {
                PaletteSetup(global.p2PaletteID, this.selectedCharacter);
            }
            startingOpacity = setStartingOpacity;
            fadeSpeed = setFadeSpeed;
            initialized = true;
        }
    }
    speedTrailTimer++;
}
```

It takes three parameters:

- setStartingOpacity sets the opacity of the sprite once you create the instance. This value should be between 0 and 1.

- setFadeSpeed sets the speed at which the sprite fades away. The sprite fades every frame, so it would be a good idea to set the fade to a very small number like 0.02.
- interval is how often a new sprite is created for the speed trail. Whenever speedTrailTimer gets bigger than this number, a new sprite is created and the timer resets.

You can use this function in any state or any part of oPlayerController/Step to create speed trails.

IMPORTANT NOTE: This function only works with the player characters. If you want to create speed trails for things like projectiles, a new system will need to be created for them.